

IDC DOCUMENTATION

**Subscription
Subsystem
Software
User Manual**



Notice

This document was published September 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors

David H. Salzberg, Science Applications International Corporation
Jan F. Wüster, Science Applications International Corporation

Trademarks

BEA TUXEDO is a registered trademark of BEA Systems, Inc.
ORACLE is a registered trademark of Oracle Corporation.
PL/SQL is a trademark of Oracle Corporation.
SAIC is a trademark of Science Applications International Corporation.
SQL*Plus is a registered trademark of Oracle Corporation.
UNIX is a registered trademark of UNIX System Labs, Inc.

Ordering Information

The ordering number for this document is SAIC-01/3029.

This document is cited within other IDC documents as [IDC6.5.21].

Subscription Subsystem Software User Manual

CONTENTS

<u>About this Document</u>	i
■ <u>PURPOSE</u>	ii
■ <u>SCOPE</u>	ii
■ <u>AUDIENCE</u>	ii
■ <u>RELATED INFORMATION</u>	iii
■ <u>USING THIS DOCUMENT</u>	iii
<u>Conventions</u>	iv
<u>Chapter 1: Introduction</u>	1
■ <u>SOFTWARE OVERVIEW</u>	2
■ <u>STATUS OF DEVELOPMENT</u>	2
■ <u>FUNCTIONALITY</u>	5
<u>Features and Capabilities</u>	6
<u>Performance Characteristics</u>	8
<u>Related Tools</u>	9
■ <u>INVENTORY</u>	9
■ <u>ENVIRONMENT AND STATES OF OPERATION</u>	11
<u>Software and Hardware Environment</u>	11
<u>Normal Operational State</u>	12
<u>Contingencies/Alternate States of Operation</u>	12
<u>Chapter 2: Operational Procedures</u>	15
■ <u>SOFTWARE STARTUP</u>	16
■ <u>SOFTWARE SHUTDOWN</u>	18
<u>Shutting Down SubsProcess</u>	18
<u>Disabling Acceptance of New Subscriptions</u>	19
<u>Disabling Registering of New Products</u>	21

■ <u>BASIC PROCEDURES</u>	22
Adding New Users	22
Making Users Unauthorized	22
Deactivating User Subscriptions	23
Obtaining Help	24
■ <u>ADVANCED PROCEDURES</u>	24
Configuring the Software	24
Tracking Subscriptions	26
Rolling-back REB Subscriptions	30
Adjusting Product Delays	32
Adding New Fileproducts	34
Configuring Signing of Subscriptions	36
■ <u>MAINTENANCE</u>	37
Purging Log Files	37
Cleaning Up Obsolete Product Descriptions	37
■ <u>SECURITY</u>	38
Passwords	39
Marking/Storing Controlled Outputs	39
<u>Chapter 3: Troubleshooting</u>	41
■ <u>MONITORING</u>	42
Verifying Subscriptions	42
Monitoring Database Tables	43
Monitoring Addition of Rows to dataready Table	45
■ <u>INTERPRETING ERROR MESSAGES</u>	49
ParseSubs Error Messages	49
SubsProcess Error Messages	51
■ <u>SOLVING COMMON PROBLEMS</u>	53
Subscription Request Fails	53
Unclear Message from SubsProcess to Subsuser	54
Product Delivery Fails	54
One User Stops Receiving One Product	54
One User Stops Receiving All Products	55

All Users Stop Receiving One Product	55
One or More Users Do Not Receive a Product	55
Empty Fileproduct Delivered	56
Subscription is Incomplete/Defective	56
Duplicated Products Delivered	57
■ ERROR RECOVERY	57
■ REPORTING PROBLEMS	58
Chapter 4: Installation Procedures	59
■ PREPARATION	60
Obtaining Released Software	60
Hardware Mapping	60
Establishing Target Information	60
UNIX System	61
Authentication	62
■ EXECUTABLE FILES	62
■ CONFIGURATION DATA FILES	63
Global Parameter Files	63
CSC-level parameter files	66
■ TEXT FILES	68
■ DATABASE	69
Accounts	69
Tables	69
Database Triggers	73
Installing “Data Ready” Processes	74
■ INITIATING OPERATIONS	75
Initializing subsuser	76
Initializing Other Tables	76
■ VALIDATING INSTALLATION	76
References	79
Appendix: Initialization	A1
■ ADDING PRODUCTS	A2

productcriteria	A2
producttypeorigin	A3
fpdescription	A3
fileproduct	A5
■ ADDING USERS	A5
subsuser.pl	A5
Glossary	G1
Index	I1

Subscription Subsystem Software User Manual

FIGURES

<u>FIGURE 1.</u>	<u>IDC SOFTWARE CONFIGURATION HIERARCHY</u>	3
<u>FIGURE 2.</u>	<u>RELATIONSHIP OF SUBSCRIPTION SUBSYSTEM TO OTHER SOFTWARE UNITS OF DATA SERVICES CSCI</u>	4
<u>FIGURE 3.</u>	<u>ENTITY RELATIONSHIPS OF SUBSCRIPTION SUBSYSTEM TABLES</u>	71

Subscription Subsystem Software User Manual

TABLES

TABLE I:	DATA FLOW SYMBOLS	v
TABLE II:	TYPOGRAPHICAL CONVENTIONS	v
TABLE III:	ENTITY-RELATIONSHIP SYMBOLS	vi
TABLE IV:	TECHNICAL TERMS	vi
TABLE 1:	SUBSCRIPTION SUBSYSTEM INVENTORY	9
TABLE 2:	FILEPRODUCT TO DATAREADY MAPPING	36
TABLE 3:	LOGS AND DATAREADY ROWS	48
TABLE 4:	SUBSCRIPTION SUBSYSTEM DATABASE TABLES	69

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- [Purpose](#)
- [Scope](#)
- [Audience](#)
- [Related Information](#)
- [Using this Document](#)

About this Document

PURPOSE

This document describes how to use the Subscription Subsystem software of the International Data Centre (IDC). The software is a computer software component (CSC) of the Data Services computer software configuration item (CSCI) and is identified as follows:

Title: Subscription Subsystem

SCOPE

The manual includes instructions for setting up the software, using its features, and basic troubleshooting. This document does not describe the software's design or requirements. These topics are described in sources cited in "[Related Information](#)." Also, the manual does not contain instructions on how to interact with the Subscription Subsystem as an authorized remote user (subsuser). Such instructions are given in the documentation volume on *Formats and Protocols for Messages* [\[IDC3.4.1Rev2\]](#). Finally, the manual assumes basic familiarity with IDC software and hardware and with the UNIX operating system.

AUDIENCE

This document is intended for the first-time or occasional user of the software. However, more experienced users may find certain sections useful as a reference. A typical user would be an IDC processing engineer or user services staff member. The document is not addressed to subsusers of the Subscription Subsystem.

RELATED INFORMATION

The following documents complement this document:

- *Subscription Subsystem* (Software Design Document) [\[IDC7.4.4\]](#)
- *Message Subsystem Software Users Manual* [\[IDC6.5.19\]](#)
- *Message Subsystem* (Software Design Document) [\[IDC7.4.2\]](#)

See [“References” on page 79](#) for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the Subscription Subsystem software:

- *fpstacap*
- *ParseSubs*
- *SubsProcess*
- *write_fp*

USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. It is part of the Technical Instructions category, which provides guidance for installing, operating, and maintaining the IDC systems. This document is organized as follows:

- [Chapter 1: Introduction](#)
This chapter provides an overview of the software’s capabilities, development, and operating environment.
- [Chapter 2: Operational Procedures](#)
This chapter describes how to use the software and includes detailed procedures for startup and shutdown, basic and advanced features, security, and maintenance. This chapter also provides instructions for creating new subscription products.

▼ About this Document

- [Chapter 3: Troubleshooting](#)

This chapter describes how to identify and correct common problems related to the software.

- [Chapter 4: Installation Procedures](#)

This chapter describes first how to prepare for installing the software, then how to install the executable files, configuration data files, and database elements. It also describes how to initiate operation and how to validate the installation.

- [References](#)

This section lists the sources cited in this document.

- [Appendix: Initialization](#)

This appendix provides SQL statements and a Perl script as examples of tools that can be used to initialize database tables, in case the Subscription Subsystem is installed from scratch or these tables have been truncated or dropped.

- [Glossary](#)

This section defines the terms, abbreviations, and acronyms used in this document.

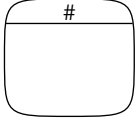

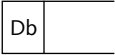

- [Index](#)

This section lists topics and features provided in this document along with page numbers for reference.

Conventions

This document uses a variety of conventions, which are described in the following tables. [Table I](#) shows the conventions for data flow diagrams. [Table II](#) lists typographical conventions. [Table III](#) lists entity-relationship symbols. [Table IV](#) explains certain technical terms that are not part of the standard Glossary, which is located at the end of this document.

TABLE I: DATA FLOW SYMBOLS

Description	Symbol ¹
process	
external source or sink of data	
data store Db = database store	
data flow	

1. Most symbols in this table are based on Gane-Sarson conventions [\[Gan79\]](#).

TABLE II: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	bold	dataready
database table and attribute, when written in the dot notation		prodtrack.status
database attributes	<i>italics</i>	<i>status</i>
processes, software units, and libraries		<i>ParseSubs</i>
user-defined arguments and vari- ables used in parameter (par) files or program command lines		<i>ParseSubs msgid=nnn</i>
titles of documents		<i>Subscription Subsystem</i>
computer code and output	courier	[fatal]: AutoDRM Killed
filenames, directories, and web- sites		SubsProcess.par
text that should be typed exactly as shown		select * from prodtrack where status='FAILED';

▼ About this Document

TABLE III: ENTITY-RELATIONSHIP SYMBOLS


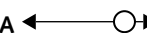








Description	Symbol														
One A maps to one B.	A  B														
One A maps to zero or one B.	A  B														
One A maps to many Bs.	A  B														
One A maps to zero or many Bs.	A  B														
database table	<table border="1"> <tr> <td colspan="2">tablename</td></tr> <tr> <td></td><td>primary key</td></tr> <tr> <td></td><td>foreign key</td></tr> <tr> <td colspan="2">attribute 1</td></tr> <tr> <td colspan="2">attribute 2</td></tr> <tr> <td colspan="2">...</td></tr> <tr> <td colspan="2">attribute n</td></tr> </table>	tablename			primary key		foreign key	attribute 1		attribute 2		...		attribute n	
tablename															
	primary key														
	foreign key														
attribute 1															
attribute 2															
...															
attribute n															

TABLE IV: TECHNICAL TERMS

Term	Description
instance	An instance describes a running computer program. An individual program may have multiple instances on one or more host computers.
subscription	A standing order for a standard or customized IDC product to be sent automatically to the (subs)user either at regular intervals or as soon as the product becomes available.
subsuser	A user authorized to register and receive subscriptions.
subscribe	To register a subscription by sending a Subscription Request Message.
unsubscribe	To cancel a subscription by sending a Subscription Request Message containing an unsubscribe line.
subscriber	A subsuser who has placed one or several subscriptions.
fileproduct	An IDC product that is stored internally in the form of a flat file.

TABLE IV: TECHNICAL TERMS (CONTINUED)

Term	Description
database product	An IDC product that is stored internally in the database and needs to be reformatted each time it is requested by a user.
cd	To use the UNIX command <code>cd</code> to change the current working directory.
grep	To use the UNIX command <code>grep</code> to search for the occurrence of a string or regular expression in text files.
fill a subscription	To assemble and ship an IDC product in response to a subscription.
wedge	To remain in the process table, but not perform its function (of a UNIX process).

Chapter 1: Introduction

This chapter provides a general description of the software and includes the following topics:

- [Software Overview](#)
- [Status of Development](#)
- [Functionality](#)
- [Inventory](#)
- [Environment and States of Operation](#)

SOFTWARE OVERVIEW

STATUS OF DEVELOPMENT

The Subscription Subsystem supports instant delivery of products (that is, delivery as soon as the data are available) including fileproducts (that is, products that exist in the form of flat files) and database products (which are created from the database on demand). It also allows subscriptions to be defined for subsets of available database products. Subsets can be based on most attributes (including automatic and associated arrivals and bulletin-associated waveforms) that are recognized by the IMS1.0 message protocol [\[IDC3.4.1Rev2\]](#). The Subscription Subsystem also adds the ability to track delivery of the products.

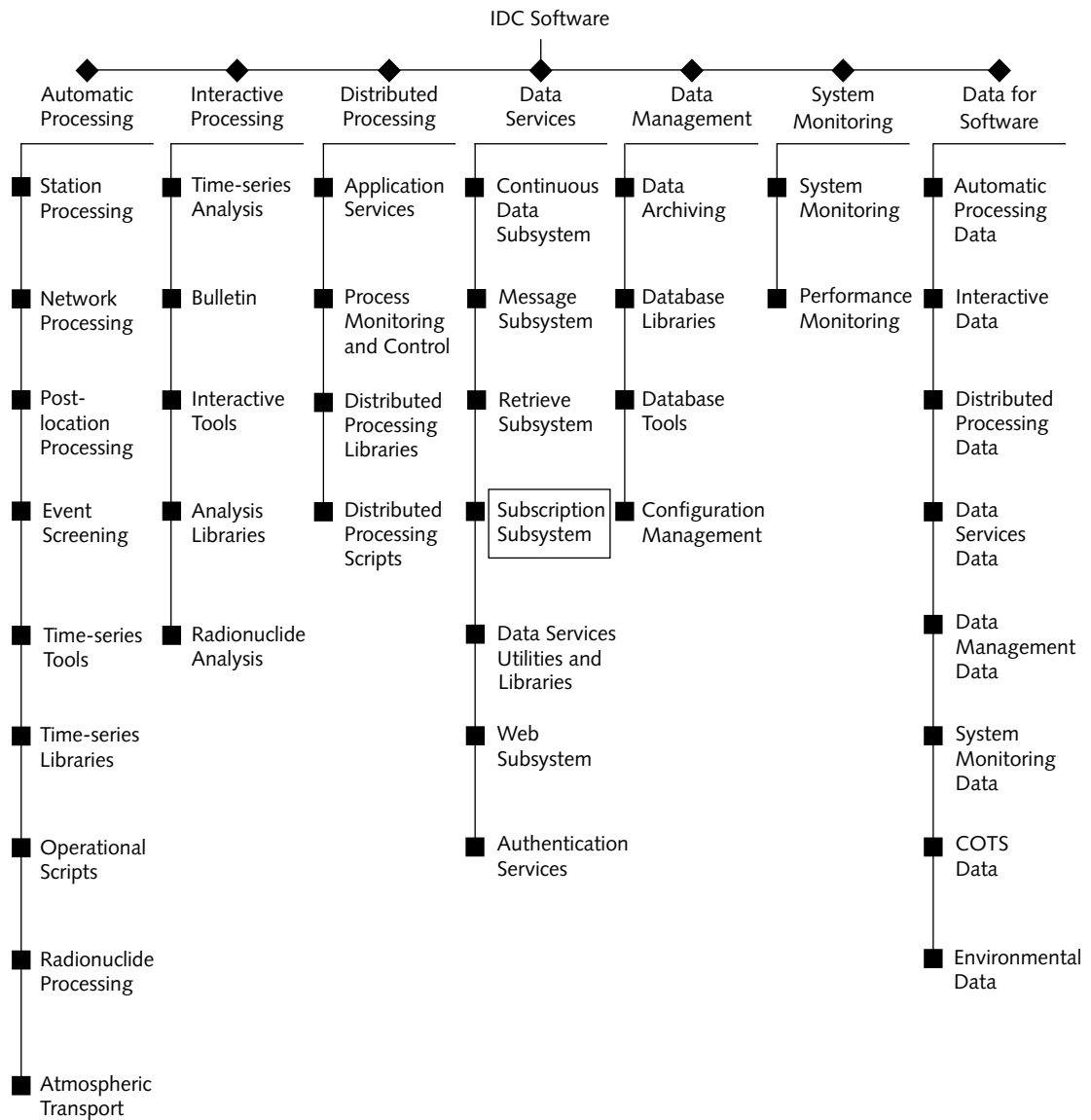


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

▼ Introduction

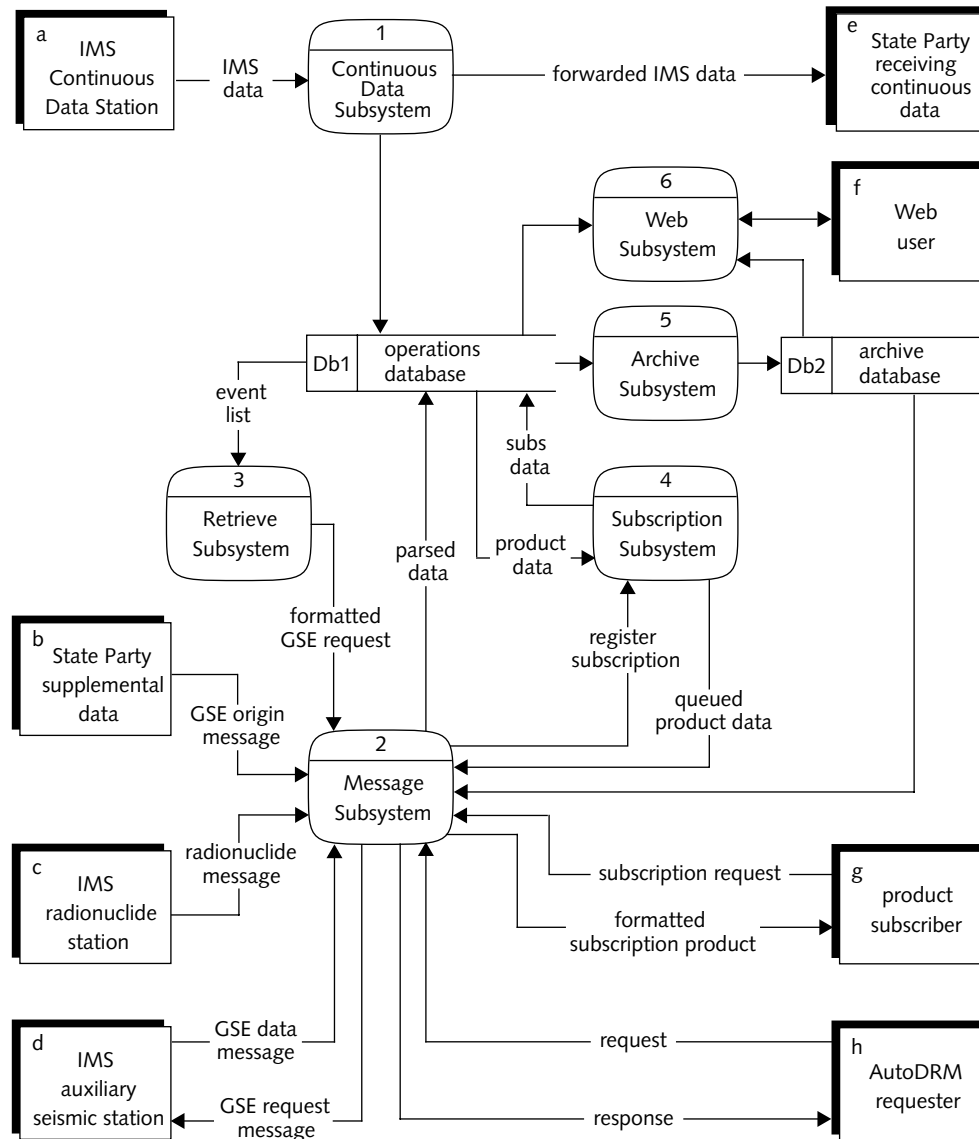


FIGURE 2. RELATIONSHIP OF SUBSCRIPTION SUBSYSTEM TO OTHER SOFTWARE UNITS OF DATA SERVICES CSCI

FUNCTIONALITY

The Subscription Subsystem enables authorized users to place and manage orders for standard and customized IDC products and to have the products delivered on a routine basis, or alternatively, as quickly as they become available. As such, it implements the functionality described in the Introduction to “Chapter 3: Subscription Messages” in [\[IDC3.4.1Rev2\]](#). See this reference for an enumeration of the available subscription products, subscription management functions, and the corresponding syntax for incoming and outgoing messages.

Requests for registering new subscriptions or managing existing subscriptions are sent by email in IMS1.0 formatted subscription request messages.¹ Upon receiving such messages from authorized users, the Subscription Subsystem sets up a subscription and stores it in the database.

Subsequently, when data become available for distribution, the Subscription Subsystem automatically determines which products or standing orders need the data and sends corresponding instructions to the Message Subsystem. The Message Subsystem completes the order and mails the products (or product announcement if the file is too large) in the form of IMS1.0 data messages to the subscribers. The Subscription Subsystem tracks delivery of the product; this auditing supports recovery after system failures using an interactive SQL*Plus session.

As described above, the Subscription Subsystem interfaces tightly with both the incoming and outgoing branches of the Message Subsystem and cannot be operated without a functioning Message Subsystem in its environment.

Users of the Subscription Subsystem must have access to a Simple Mail Transfer Protocol (SMTP) mail agent. A file transfer protocol (FTP) application or a web-based retrieval mechanism is needed to obtain voluminous products. It is also advisable that the recipient of the subscription has software to parse the delivered data.

A brief description of the main components of the Subscription Subsystem follows:

-
1. A Web interface has been designed, as mentioned in the software design document of the Subscription Subsystem [\[IDC7.4.4\]](#), but has not actually been implemented.

▼ Introduction

<i>fpstacap</i>	Creates fileproduct entries for each station in a station status report.
<i>ParseSubs</i>	Parses subscription request messages and registers subscriptions in the database for authorized users.
<i>SubsProcess</i>	Queues, processes, and formats Subscription products.
<i>write_fp</i>	Registers the availability of fileproducts in the database.

ParseSubs is started by *MessageGet*, a component of the Message Subsystem, when a Subscription Request message has been received. *SubsProcess* is invoked from the script *keep_subs_alive.sh* at startup or from *cron*; it should always be running while the Subscription Subsystem is activated. *write_fp* can be called by any process that creates a fileproduct to make the completion of this fileproduct known to the Subscription Subsystem. *fpstacap* is called by *run_stacap* (a script that calls the utility *stacap*, which creates station reports), which in turn is invoked by *cron*.

Refer to the software design description of the Subscription Subsystem [\[IDC7.4.4\]](#) for a description of the relationship of the software's functions with the interfacing systems.

Features and Capabilities

Incoming IMS1.0 subscription request messages are recognized by the Message Subsystem process *MessageGet*, which invokes *ParseSubs* to parse them.

ParseSubs first verifies that the subscription request message comes from an authorized user of the Subscription Subsystem by attempting to look up the email originator (from the **msgdisc.isrc** value) in the database table **subsuser** (**subsuser.username** and **subsuser.domain**).² This check is in addition to a potential message authentication by the Message Subsystem and the requirement for an entry in the Message Subsystem's **datauser** table. If the lookup fails, the subscrip-

2. The domain is modified from the domain in the email; if there is more than one "dot" in the address, *ParseSubs* assumes that the email is of the form *machine.domain.network*, and it converts it to what it thinks is *domain.network*.

tion request is considered unauthorized; the message in file `$(CMS_CONFIG)/app_config/messages/ParseSubs/NotAuthorized` is emailed to the originator of the subscription request, and the incident is recorded in the *ParseSubs* log file. In addition, the *msgdisc.status* is updated to `Failed`. Authorized messages are accepted for parsing, the first step of which is syntax checking. If syntax errors are found, *ParseSubs* notifies the user of the problem encountered by dispatching an email with an `IMS1.0 ERROR_LOG` message back to the user, including an attempt to diagnose the error. Syntactically correct subscription requests cause *ParseSubs* to take appropriate action to register the subscription:

- *ParseSubs* initiates a new subscription by creating appropriate entries in the database table **subs**; and if the subscription request introduces new bounding conditions it also creates entries in tables **productcriteria**, **productypesta**, **productypeorigin**, or **productypeevsc**. Rows are added to the same tables when establishing a national bulletin product.
- *ParseSubs* modifies an existing subscription by unsubscribing to the product and adding a subscription to a different product (it may be new or preexisting). The old product definition is not deleted because other users may be subscribed to the same product.
- Information on existing subscriptions is provided by selecting from these tables: **subs**, **productcriteria**, **productypesta**, **productypeorigin**, and **productypeevsc**.
- *ParseSubs* resubmits a previous delivery of an existing subscription by updating the *msgdest.status* value to `PENDING`.
- *ParseSubs* terminates an existing subscription by updating **subs.status** to `i` (inactive).

In all cases the user is notified of the action taken with an email containing an `IMS1.0 LOG` message.

Various IDC processes insert a row into the **dataready** table each time they complete a certain processing step to indicate that the processing results just created are now available for distribution. In the case of database products this happens directly with the help of Perl scripts (*RebDone_into_DataReady* executed within the

▼ Introduction

REB pipeline) and SQL scripts (*Origin_into_DataReady.sql*, *Orid_into_DataReady.sql*, and *Sta_into_DataReady.sql* executed within the SEL1, SEL2, and SEL3 pipelines). In contrast, fileproducts are first registered by inserting a row into the **fileproduct** table (by calls to *fpstacap* in the case of station capability reports and by calls to *write_fp* in all other cases), which in turn causes a row to be inserted into **dataready** via the database trigger *fileproduct_to_dataready*.

The PL/SQL module *DataReady_to_ProdTrack* makes the logical connection between completed processing steps and the creation of IDC products. It gets triggered by each insertion of a new row (standing for a newly completed processing step) into the **dataready** table, and writes one row into the **prodtrack** table for each product that is to be generated from these data. This may be a one-to-many relationship, as in the case where one underlying bulletin (for example, the Reviewed Event Bulletin [REB]) forms the basis for the creation of multiple products derived by different constraints on geographical location, magnitude, or other criteria.

SubsProcess oversees the creation and shipping of the subscribed data products. It periodically searches the **prodtrack** table for rows with **prodtrack.status** = new, then obtains the definition and constraints of the desired product by selecting from the product tables (**productcriteria** and **producttypesta**, **producttypeorigin**, or **producttypeevsc**), constructs an IMS1.0 request message for the product, invokes an instance of *AutoDRM*, and passes the message to this instance for further handling of the request.

AutoDRM, a component of the Message Subsystem, reads the request messages, retrieves the data, formats the products and writes them to files, which are subsequently shipped by the outgoing branch of the Message Subsystem.

Performance Characteristics

So far the Subscription Subsystem has not been systematically stress-tested. In June 2001, 152 subscriptions were active for 37 distinct products at the Prototype International Data Centre (PIDC) with an average total of 7.8 MB/day shipped (with peak values of 14 MB/day on days when REBs are produced). The response time to set up or modify a subscription for an authorized user is typically on the

order of seconds up to a few minutes. The mean delay between the completion of a processing step (row inserted in **dataready**) and the shipping of associated subscribed products is mainly driven by the loop-time of *SubsProcess* (parameter *loop*, which defaults to 60 seconds).

The module *DataReady_to_ProdTrack* is activated very often, due to the fact that the completion of each Detection and Feature Extraction (DFX) interval results in one row to be inserted into the **dataready** table to support immediate subscriptions to the Standard List of Signal Detections (SLSD) product. The module has recently been optimized for efficiency, as there were indications during episodes of database slowdown that it was contributing to the problem. However, cause and effect could not be conclusively established in these cases.

No failures are expected to occur in the normal course of processing, so an expected error rate cannot be provided. Any failure that does occur should be investigated and documented (see [“Chapter 3: Troubleshooting” on page 41](#)).

Related Tools

A script *keep_subs_alive.sh* is delivered with the Subscription Subsystem and is discussed under [“Functionality” on page 5](#) and in [“Chapter 2: Operational Procedures” on page 15](#)

INVENTORY

The Subscription Subsystem includes the components listed in [Table 1](#).

TABLE 1: SUBSCRIPTION SUBSYSTEM INVENTORY¹

Item	Type
<i>fpstacap</i>	executable binary
<i>ParseSubs</i>	executable binary
<i>SubsProcess</i>	executable binary
<i>write_fp</i>	executable binary

▼ Introduction

TABLE 1: SUBSCRIPTION SUBSYSTEM INVENTORY¹ (CONTINUED)

Item	Type
AutoDRM_subs.par	parameter file (owned by the Message Subsystem, but with entries related to the Subscription Subsystem)
MessageGet.par	parameter file (owned by the Message Subsystem, but with entries related to the Subscription Subsystem)
msgs.par	parameter file
ParseData.par	parameter file (owned by the Message Subsystem, but with entries related to the Subscription Subsystem)
ParseSubs.par	parameter file
shared.par	parameter file
stacap.par	parameter file
SubsProcess.par	parameter file
write_fp.par	parameter file
NotAuthorized	text file
libfileproduct	library
RebDone_into_DataReady	Perl script
dataready	database table (owned)
fileproduct	database table (owned)
fpdescription	database table (owned)
prodtrack	database table (owned)
productcriteria	database table (owned)
producttypeevsc ²	database table (owned)
producttypeorigin	database table (owned)
producttypepesta	database table (owned)
subs	database table (owned)
subsuser	database table (owned)

TABLE 1: SUBSCRIPTION SUBSYSTEM INVENTORY¹ (CONTINUED)

Item	Type
msgdisc	database table (used)
datauser	database table (used)
<i>DataReady_to_ProdTrack.sql</i>	SQL script
<i>fileproduct_to_dataready.sql</i>	SQL script
<i>Orid_into_DataReady.sql</i>	SQL script
<i>Origin_into_DataReady.sql</i>	SQL script
<i>Sta_into_DataReady.sql</i>	SQL script
<i>keep_subs_alive.sh</i>	shell script

1. For configuration of database accounts see [\[IDC5.1.3Rev0.1\]](#). For configuration of database tables see [\[IDC5.1.1Rev2\]](#). For configuration of parameter files see "[Chapter 4: Installation Procedures](#)" on page 59 of this document and [\[IDC6.2.4\]](#).
2. This table was introduced after the completion of [\[IDC7.4.4\]](#) and is thus not described in that document.

ENVIRONMENT AND STATES OF OPERATION

Software and Hardware Environment

The Subscription Subsystem is designed to be closely integrated with the Message Subsystem, which must be installed properly for the Subscription Subsystem to function.

The Subscription Subsystem requires a database constructed according to the IDC Database Schema [\[IDC5.1.1Rev2\]](#). Its software modules operate on this database through a Generic Database Interface Library (*libgdi*), which must be accessible at run time. The Subscription Subsystem needs to access the database account that is used by the Continuous Data Subsystem and by detection and station processing.

▼ Introduction

This account is called IDCX. The configuration of tables owned by the Subscription Subsystem is described in [“Chapter 4: Installation Procedures” on page 59](#) of this document.

The Subscription Subsystem does not require any additions to its hardware or software environment over and above what is required to successfully install and operate the Message Subsystem. Although it is not required, the Subscription Subsystem is designed to run on the same host as the Message Subsystem. Refer to Chapter 1 of [\[IDC6.5.19\]](#) for the minimum requirements of the Message Subsystem. In particular, access to UNIX mail services is essential for the Subscription Subsystem to function, and subscribers need access to FTP services to retrieve large volume subscriptions.

The Subscription Subsystem does not rely on any aspect of the Distributed Applications Control System (DACS); however, items that trigger the subscription delivery process are often stop-hooks to DACS-controlled processes.

Normal Operational State

The Subscription Subsystem is designed to operate continuously. *ParseSubs* is invoked by *MessageGet*, a component of the Message Subsystem, each time a Subscription Request Message is received. The whole interlocking mechanism of product generation is triggered by various steps of automatic and interactive processing. *SubsProcess* is started by the operator after system installation and subsequently after each system boot by *cron* through the *keep_subs_alive.sh* script. This script, intended to be called regularly by *cron*, will restart *SubsProcess*, if it does not find an instance of it in the host's UNIX process table.

Contingencies/Alternate States of Operation

You can run *ParseSubs* from a command line for testing purposes. Use the following syntax (assuming that *ParseSubs.par* is in your current directory):

```
ParseSubs par=ParseSubs.par msgid=nnn
```

You can use *write_fp* on a command line and in scripts to insert lines into the **fileproduct** table as follows:

```
writefp par=writefp.par dir=directory dfile=filename \  
time=epoch_start_time endtime=epoch_end_time
```

You can use *fpstacap* when a flat file with a station report is available:

```
fpstacap par=fpstacap.par dir=directory dfile=filename
```

You can insert rows into the **dataready** table by using SQL queries; this will trigger *DataReady_to_ProdTrack* and begin product generation in the same way as an automatically inserted new row in **dataready**.

You can make *SubsProcess* reprocess a subscription by updating **prodtrack.status** to **NEW**.

Chapter 2: Operational Procedures

This chapter provides instructions for using the software and includes the following topics:

- [Software Startup](#)
- [Software Shutdown](#)
- [Basic Procedures](#)
- [Advanced Procedures](#)
- [Maintenance](#)
- [Security](#)

Chapter 2: Operational Procedures

SOFTWARE STARTUP

The Subscription Subsystem consists of three functional branches: parsing subscription requests, registering data products, and filling subscriptions.

The process *ParseSubs* parses and registers incoming subscription requests. *ParseSubs* is spawned by the process *MessageGet* (a component of the Message Subsystem) each time a Subscription Request Message is received. You do not need to perform any action to start *ParseSubs*.

A combination of PL/SQL scripts and database triggers registers available IDC products and prepares the requests for shipment to subscribers. Product registration is driven by processes that complete the data processing steps, which make the respective IDC products available. You do not need to perform any action to start this branch.

Components of the Message Subsystem actually fill the subscriptions by assembling and shipping the products, but the process *SubsProcess* initiates this procedure. *SubsProcess* is intended to continuously run in the background like a daemon. It needs to be started either by the user or by an automatic process. The simplest way to start *SubsProcess* is from the command line:

1. Log onto the machine intended to run the Message Subsystem and the Subscription Subsystem (MSGHOST) as the functional user owning the processes of these subsystems (AUTO).

2. Check to see if *SubsProcess* is running:

```
% ps -fe | grep SubsProcess
```

If *SubsProcess* is running, expect output similar to the following:

```
auto 3225 1 0 13:36:40 ? 1:11 /cmss/rel/bin/SubsProcess
```

In this case, the PID is 3225.

3. Send the UNIX SIGTERM signal to the process:

```
$ kill 3225
```

4. Confirm that *SubsProcess* has died by repeating Step 2. If it has not, repeat Step 3.

5. If SIGTERM is ineffective, use SIGKILL:

```
$ kill -9 3225
```

6. Repeat Step 2 to show that *SubsProcess* has now died.

7. Issue the following command on the command line:

```
% /cmss/rel/bin/SubsProcess \  
par=/cmss/config/app_config/messages/SubsProcess/SubsProcess.par
```

8. Check if *SubsProcess* is running by querying the UNIX process table:

```
$ ps -fe | grep SubsProcess
```

Expect output similar to the following:

```
auto 3239 1 0 13:36:40 ? 1:11 /cmss/rel/bin/SubsProcess
```

A shell script *keep_subs_alive.sh* is provided to facilitate the startup of *SubsProcess*. It is intended to be run regularly (for example every 20 minutes) out of *cron*. It checks first if an instance of *SubsProcess* is already running on the machine in question and launches one if that is not the case.

To set up such a cron job, follow the established procedures at your site. The following procedure is provided as an example:

1. Log onto the machine intended to run the Message Subsystem and the Subscription Subsystem (MSGHOST) as the functional user owning the processes of these subsystems (AUTO).
2. Change directory to the location of preformed crontab entries, for example:

```
$ cd /cmss/config/host_config/crontab
```

▼ Operational Procedures

3. Substitute symbolic parameters in the `crontab` entry file:

```
cat auto-crontab-subs_keep_alive.txt | \
solvepar > /tmp/auto-crontab-subs_keep_alive.txt.txt
```

4. Edit the `crontab` entry file if necessary:

```
$ vi auto-crontab-subs_keep_alive.txt
```

Expect output similar to the following:

```
#
#-----Monitor and restart Subscription Subsystem applications
#           that have exited
#
#0,20,40 * * * * ( /usr/bin/env '/usr/bin/cat
/cmss/config/system_specs/env/global.shenv'
/cmss/scripts/bin/keep_subs_alive >/dev/null 2>&1)
```

5. Using a cut-and-paste operation with the `vi` editor, add these lines (without the `#` comment character on the line beginning with `#0,20,40`) to the `crontab` file (owned by `AUTO` on `MSGHOST`):

```
$ crontab -e
```

SubsProcess automatically starts at the next full 20 minutes. You can check the successful launch with the UNIX command `ps` (as described above).

SOFTWARE SHUTDOWN

Shutting Down SubsProcess

The only component of the Subscription Subsystem that is running continuously is *SubsProcess*. Shutting it down will prevent subscriptions from being filled. However, parsing and registering of new subscriptions will still be possible after the shutdown, and the registering of available products will also proceed.

To shut down *SubsProcess*, kill the process as follows:

1. Log onto the machine `MSGHOST` as user `AUTO`.

2. Comment the start script for *SubsProcess* out of crontab:

```
$ crontab -e
```

Place the comment sign (#) in front of the line that references *keep_subs_alive*.

3. Obtain the process identifier (PID) from the UNIX process table:

```
$ ps -fe | grep SubsProcess
```

Expect output similar to the following:

```
auto 3239 1 0 13:36:40 ? 1:11 /cmss/rel/bin/SubsProcess
```

In this case, the PID is 3239.

4. Send the UNIX SIGTERM signal to the process:

```
$ kill 3239
```

5. Confirm that *SubsProcess* has died by repeating Step 3. If it has not, repeat Step 4.

6. If SIGTERM is ineffective, use SIGKILL:

```
$ kill -9 3239
```

7. Repeat Step 3 to show that *SubsProcess* has now died.

Disabling Acceptance of New Subscriptions

The following short-term procedure blocks the subscription queue:

1. Select any row from the **msgdisc** table with *msgtype* = SUBSCRIPTION, for example, the latest one that has been completed, by using the following SQL command:

```
SQL> select max(msgid) from msgdisc
      2  where msgtype='SUBSCRIPTION' and status='DONE';
```

```
MAX(MSGID)
```

```
-----
```

```
2015951
```

▼ Operational Procedures

2. Update `msgdisc.status` of this message to `RUNNING` to block *MessageGet* from accepting any new subscription request messages by using the following SQL command:

```
SQL> update msgdisc set status='RUNNING' where msgid=2015951;  
1 row updated  
SQL> commit;
```

3. To enable acceptance of subscriptions again, reverse the update.

The following long-term procedure reduces the number of concurrently allowed subscription request messages to 0.

1. Edit the *MessageGet* par file.

```
$ cd $(CMS_CONFIG)/app_config/messages/MessageGet  
$ vi MessageGet.par
```
2. Find the number of the message-type for subscription request messages; in the current configuration it is as follows:

```
message-type-6=subscription
```
3. Reduce the parameter *max-running* for message type number 6 to 0:

```
max-running-6=0
```
4. Shut down and restart the process *MessageGet* on `MSGHOST` as `AUTO` (see Chapter 2 in the *Message Subsystem Software User Manual* [\[IDC6.5.19\]](#)).

Disabling Registering of New Products

1. Disable the triggers *fileproduct_to_dataready* and *DataReady_to_ProdTrack* by using the following SQL commands:

```
SQL> alter trigger fileproduct_to_dataready disable;
SQL> alter trigger DataReady_to_ProdTrack disable;
SQL> commit;
```

Caution: THIS ACTION WILL HAVE THE EFFECT THAT PRODUCTS THAT BECOME READY FOR SHIPPING WHILE THE TRIGGER IS DISABLED WILL NEVER BE REGISTERED BY THE SUBSCRIPTION SUBSYSTEM AND CONSEQUENTLY WILL NEVER BE SHIPPED AS SUBSCRIPTIONS.

To enable the triggers again, use the following SQL commands:

```
SQL> alter trigger fileproduct_to_dataready enable;
SQL> alter trigger DataReady_to_ProdTrack enable;
SQL> commit;
```

2. "Comment out" the subscription-related stop-hooks in the following files (paying attention to version control where required):

```
$(CMS_CONFIG)/app_config/distributed/tuxshell/...
.../sel1/tuxshell-WE-sel1.par
.../sel2/tuxshell-WE-sel2.par
.../sel3/tuxshell-Beamer.par
.../reb/tuxshell-ol2or.par
.../detpro/tuxshell-StaPro.par
.../segarch/tuxshell-doday.par
```

3. "Comment out" the boolean parameter *enable-waveform-sub* in `$(CMS_CONFIG)/app_config/messages/ParseData/ParseData.par` to disable the registering of newly arriving auxiliary data as available subscription products.
4. There is currently no parameter to selectively disable the registration of Event Screening products. Writing to **dataready** can only be disabled in conjunction with disabling database writes in general and should not be done.

▼ Operational Procedures

BASIC PROCEDURES

The Subscription Subsystem requires no routine operator interventions with the exception of adding and removing authorized users and deactivating the subscriptions of removed users if desired. These actions are explained below.

Adding New Users

To add a new user, use an SQL query to manually insert a new row containing this user's attributes into the **subsuser** table. An example is given in [Chapter 4: Installation Procedures](#) in ["Initiating Operations" on page 75](#). It is advisable to use a script for this purpose; an example of a suitable script (*subsuser.pl*) is provided in ["Appendix: Initialization" on page A1](#).

Making Users Unauthorized

To make an existing user unauthorized to access the Subscription Subsystem remove the corresponding row from the **subsuser** table. The following procedure assumes that the user is Jane.DOE@nirwana.org:

1. Connect to the appropriate database account (IDCX):

```
$ sqlwrap IDCXDB
```

2. Find the *userid* of the user in question:

```
SQL> select userid, username, status from subsuser
      2  where upper(username)='JANE.DOE'
      3    and upper(domain)='NIRWANA.ORG';
```

Expect one and only one row; note the *userid*:

USERID	USERNAME	STATUS
1581	jane.doe	a

3. Delete the row by entering the following:

```
SQL> delete from subsuser where userid=1581;  
1 row deleted  
SQL> commit;
```

After this procedure, subscription request messages from user Jane.Doe will no longer be processed; instead, the `NotAuthorized` message will be returned. However, existing subscriptions for Jane.Doe will still be filled. If that is to be prevented, they must be deactivated (see next subsection) or deleted completely:

4. Delete all subscriptions with `userid = 1581`;

```
SQL> delete from subs where userid=1581;  
6 rows deleted  
SQL> commit;
```

Note: This procedure does not leave a record of Jane's subscriptions. In case a colleague of hers is to assume her role and inherit her subscriptions, it may be more practical to leave the subscriptions intact.

Deactivating User Subscriptions

To stop all subscriptions for a particular user from being distributed, change their status in the **subs** table from `a` (active) to `i` (inactive):

1. Open an SQL session in the IDCX account, and find **subsuser.userid** as explained in steps 1 and 2 of the previous procedure.
2. Assuming the `userid` is 1581 for this example, deactivate all subscriptions:

```
SQL> update subs set status='i' where userid=1581;  
6 rows updated  
SQL> commit;
```

Alternatively, the **subs.offdate** can be updated to the current `sysdate`, which will have the effect of letting all subscriptions of the user expire:

```
SQL> update subs set offdate=sysdate where userid=1581;  
6 rows updated  
SQL> commit;
```

▼ Operational Procedures

This has the advantage of leaving a record in the database when the subscription was disabled. If it subsequently is to be enabled again, a new **subs** row with **subs.ondate=sysdate** can be created manually.

Obtaining Help

Help for the operator is available from the man pages *ParseSubs(1)*, *SubsProcess(1)*, *write_fp(1)*, *fpstacap(1)*, from this manual and from the following IDC documents:

- *Subscription Subsystem* (Software Design Document) [\[IDC7.4.4\]](#)
- *Message Subsystem Software Users Manual* [\[IDC6.5.19\]](#)
- *Message Subsystem* (Software Design Document) [\[IDC7.4.2\]](#)

Help for the subsuser is available from *Formats and Protocols for Messages* [\[IDC3.4.1Rev2\]](#). Non-authorized users who send a subscription request message will receive the `NotAuthorized` message, which directs them to an email address and a procedure to request authorization. In addition, if non-authorized users send informal email messages to the established email address of user services (for example, at the IDC, `user_services@CTBTO.ORG`) they will be put in contact with an operator.

ADVANCED PROCEDURES

This section provides detailed instructions for using the software's advanced features.

Configuring the Software

The Subscription Subsystem is delivered with a complete set of configuration data files. Certain parameters are site specific and may need to be adjusted. These are isolated in a few site-specific configuration data files, which are described in ["Chapter 4: Installation Procedures" on page 59](#).

Changes of configuration may become necessary as part of the installation of patch releases. You may also want to change the configuration to adjust aspects of the software's functionality.

The following portion of the configuration tree under \$(CMS_CONFIG) shows the location of the site-specific configuration data files that influence the Subscription Subsystem:

```
system_specs/process.par
system_specs/shared.par
system_specs/msgs.par
system_specs/env/global.shenv
```

The following portion of the configuration tree shows the location of the non-site-specific configuration data files that influence the Subscription Subsystem:

```
app_config/messages/AutoDRM/AutoDRM_subs.par
app_config/messages/MessageGet/MessageGet.par
app_config/messages/ParseSubs/ParseSubs.par
app_config/messages/ParseSubs/NotAuthorized
app_config/messages/SubsProcess/SubsProcess.par
app_config/misc/stacap/stacap.par
app_config/misc/stacap/write_fp/write_fp.par
```

The syntax and semantics of all parameters are described in their respective man pages.

Changes to all site-independent parameter files, except `MessageGet.par` and `SubsProcess.par`, can be made (in accordance with established procedures of configuration management) without shutting down any aspect of the system. The new values will take effect when the respective module is run the next time.

Changes to the site-dependent parameter files and to `MessageGet.par` and `SubsProcess.par` require those modules to be shut down and restarted to take effect. See sections “Software Startup” and “Software Shutdown” in Chapter 2 of the *Message Subsystem Software User Manual* [\[IDC6.5.19\]](#) and [“Software Startup” on page 16](#) and [“Software Shutdown” on page 18](#) of this document for the shut-down and startup procedures.

▼ Operational Procedures

Tracking Subscriptions

This section describes how to track products from their first registration to the record of filling subscriptions. The following procedure describes how to:

- Determine the *dataid* from the **dataready** table.
- Find the products that used the data (**prodtrack** table).
- Find the *msgids* for those products.
- Inspect the corresponding **msgdisc** row.
- Look at the file specified in the *dir/dfile* fields.
- Look at the **msgdest** row to see who actually received the subscription.

This example shows the reception of the event that occurred at 993080679 epoch time (orid=21308425).

Find the **dataready** row:

```
% /cmss/rel/bin/e2h 993080678
993080678.00000 2001171 2001/06/20 23:44:38.00000 Jun Wed

SQL> select * from idx.dataready
2  where tagname='jdate' and tablename='origin'
3  and account='reb' and tagid=2001171;
```

DATAID	TABLERNAME	ACCOUNT	TAGNAME	TAGID	TAGID2	DATA_INFO	STATUS

LDDATE							

14075995	origin	reb	jdate	2001171	-1	-	n
29-JUN-2001							

Now, find the **prodtrack** rows:

```
SQL> select prodid, dataid, msgid, status
2  from idx.prodtrack where dataid=14075995;
```

PRODID	DATAID	MSGID	STATUS

4	14075995	7956755	DONE
10	14075995	-1	NOACTIVE
150	14075995	-1	NOACTIVE
1011	14075995	-1	NOACTIVE
1012	14075995	-1	NOACTIVE
1013	14075995	-1	NOACTIVE
1016	14075995	7956756	DONE
1052	14075995	-1	NOACTIVE
1053	14075995	-1	NOACTIVE
1054	14075995	-1	NOACTIVE
1068	14075995	-1	NOACTIVE
1070	14075995	7956757	DONE
1074	14075995	-1	NOACTIVE
1080	14075995	7956758	DONE
1082	14075995	-1	NOACTIVE
1092	14075995	7956759	DONE

Five products are generated.

▼ Operational Procedures

Now, find the **msgdisc** rows:

```
SQL> select * from idx.msgdisc m, idx.prodtrack p
      where p.msgid=m.msgid and p.dataid=14075995;
```

MSGID	USERID	MSGVER	MSGTYPE	SU	EXTMSGID
INTID	INTIDTYPE	MSGSRC	ITIME	IDATE	IMETHOD
ISRC	MSIZE	STATUS	SUBJECT		
DIR		DFILE	FOFF	MFOFF	
FILEOFF	FILESIZE	SIGTYPE	VERI	COMMID	LDDATE
PRODID	DELIVID	DATAID	MSGID	STATUS	LDDATE
7956755	-1	IMS1.0	DATA	-	4
1314	delivid	GSE_IDC	993846621	2001180	-
Subscription	178199	NEW	-		
/messages/2001/180		7956755.msg	0	65	
0	178199	-	-	-1	29-JUN-2001
4	1314	14075995	7956755	DONE	29-JUN-2001
7956756	-1	IMS1.0	DATA	-	1016
703	delivid	GSE_IDC	993846634	2001180	-
Subscription	178201	NEW	-		
/messages/2001/180		7956756.msg	0	67	
0	178201	-	-	-1	29-JUN-2001
1016	703	14075995	7956756	DONE	29-JUN-2001
7956757	-1	IMS1.0	DATA	-	1070
115	delivid	GSE_IDC	993846648	2001180	-
Subscription	178201	NEW	-		
/messages/2001/180		7956757.msg	0	67	
0	178201	-	-	-1	29-JUN-2001
1070	115	14075995	7956757	DONE	29-JUN-2001
7956758	-1	IMS1.0	DATA	-	1080
68	delivid	GSE_IDC	993846657	2001180	-
Subscription	178200	NEW	-		
/messages/2001/180		7956758.msg	0	66	
0	178200	-	-	-1	29-JUN-2001
1080	68	14075995	7956758	DONE	29-JUN-2001
7956759	-1	IMS1.0	DATA	-	1092
48	delivid	GSE_IDC	993846664	2001180	-
Subscription	178200	NEW	-		
/messages/2001/180		7956759.msg	0	66	
0	178200	-	-	-1	29-JUN-2001
1092	48	14075995	7956759	DONE	29-JUN-2001

From any one of these files (specified by *dfile*), you can determine the bulletin of interest:

```
$ cat /messages/2001/180/7956759.msg
```

```
EVENT 21301306 EASTERN NEW GUINEA REG., P.N.G.
```

```

      Date      Time      Err  RMS Latitude Longitude  Smaj  Smin
      Az Depth  Err Ndef Nsta Gap  mdist  Mdist Qual   Author
OrigID
2001/06/20 23:44:38.51  1.44  0.70  -5.9789  147.6299  54.2  21.4
  112   0.0f           6   6 169  18.96  85.22 m i uk PIDC_REB
21308425
```

```

Magnitude  Err Nsta Author      OrigID
ML         4.6 0.3    1 PIDC_REB 21308425
mb         4.3 0.2    5 PIDC_REB 21308425
mb1        4.6 0.2    6 PIDC_REB 21308425
mbmle      4.1 0.2   13 PIDC_REB 21308425
mb1mx      4.4 0.1   14 PIDC_REB 21308425
Ms         3.1 0.2    1 PIDC_REB 21308425
Ms1        3.1 0.2    1 PIDC_REB 21308425
msmle      2.9 0.1    9 PIDC_REB 21308425
ms1mx      2.8 0.1    9 PIDC_REB 21308425
```

```

Sta      Dist  EvAz Phase      Time      TRes  Azim AzRes  Slow
      SRes Def   SNR      Amp  Per Qual Magnitude  ArrID
WRA      18.96 221.7 P          23:49:01.659  0.1  41.8  -3.8  11.6
      0.1 TAS  36.8          2.2  0.33 a__ ML      4.6 31758210
                                mb1      5.0
ASAR     21.99 215.5 P          23:49:35.050  0.6  55.2   7.2   9.2
      -1.3 TAS 100.6        11.5  0.44 a__ mb      4.6 31758193
                                mb1      4.9
ASAR     21.99 215.5 LR        23:58:52.133 48.7 356.8 -42.2  38.8
      2.2 ____          48.3 20.42 a__ Ms      3.1 31806760
                                Ms1      3.1
FITZ     24.61 238.8 P
```

▼ Operational Procedures

You can determine which users actually received this event:

```
SQL> select emailto from idcx.msgdest m, idcx.prodtrack p
      2 where p.msgid=m.msgid and p.dataid=14075995;
```

EMAILTO

```
-----
mason@cmr.gov
north@cmr.gov
frode@norsar.no
bowman@cmr.gov
tormod@norsar.no
wagner@multimax.com
idcrpt@rayleigh.tt.aftac.gov
reb@norsar.no
caron@cmr.gov
scatter@cmr.gov
hayward@passion.isem.smu.edu
zatlouka@cmr.gov
nrecord@cmr.gov
pidc-products@tvo.dtra.mil
bowman@cmr.gov
services@ctbto.org
salzberg@cmr.gov
```

Rolling-back REB Subscriptions

In case an REB has been submitted in error or prematurely, the following procedure prevents REB-based subscriptions from going out and resets the database to a state in which the same REB can be submitted again.¹

1. Immediately shut down *SubsProcess* (using UNIX *kill*, as described in ["Software Shutdown" on page 18](#)).
2. Let the process that migrates the Late Event Bulletin (LEB) to the REB (*mig_bull* with *par=ol2or.par*) run to completion.

1. This procedure covers only the subscription-related issues of such a cleaning operation, and in a separate action the entries in the REB account for this particular dataday must be removed.

- Obtain the *jdate* of the day for which LEB-to-REB migration has just run; say that was for July 5, and today is July 10:

```
$ today -5
994291200.000 2001186 Jul 5,2001 0:00:00.000
```

- Determine if the REB-based subscriptions were distributed. If so, inform users that these subscriptions were distributed in error and should be disregarded. First, obtain the *dataid* for the REB-based subscriptions:

```
$ sqlwrap IDCXDB
SQL> select dataid from DATAREADY
  2   where TAGNAME='jdate' and TAGID=<jdate value>
  3   and ACCOUNT='reb' and TABLENAME='origin';
```

Then, determine status of those subscriptions from **prodtrack**:

```
SQL> select * from PRODTRACK where DATAID=<dataid>;
```

Any row in which the *status* field is set to DONE stands for a subscription that has been distributed.

- Delete the row pertaining to the problematic subscription from **dataready**:

```
SQL> delete from DATAREADY where DATAID=<dataid>;
1 row deleted
SQL> commit;
```

- Roll back **productcriteria.delivid**:

```
SQL> update PRODUCTCRITERIA set DELIVID=DELIVID-1
  2   where PRODID in
  3   (select PRODID from PRODTRACK where DATAID=<dataid>);
nn rows updated
SQL> commit;
```

- Delete rows from **prodtrack**:

```
SQL> delete from PRODTRACK where DATAID=<dataid>;
mm rows deleted
SQL> commit;
SQL> quit;
```

Note: WHEN DELETING FROM AND UPDATING DATABASE TABLES, ALWAYS CHECK THAT THE NUMBER OF ROWS DELETED OR UPDATED CORRESPONDS TO THE NUMBER YOU EXPECTED. IF THERE IS A DISCREPANCY, DO NOT QUIT ORACLE; INSTEAD, ISSUE A ROLLBACK COMMAND AND INVESTIGATE.

▼ Operational Procedures

Adjusting Product Delays

“Immediate” subscriptions are released when the products become available. Products become available for subscription processing when a row in the **dataready** table is created. For database products, this row is created by SQL scripts, which are called from stop-hooks. A stop-hook is a feature of the application server *tuxshell* (a component of the DACS, see [\[IDC6.5.2Rev0.1\]](#) and [\[IDC7.3.1\]](#)) that executes a command after the successful execution of the child application.

In some cases an artificial delay is introduced by means of a lookback parameter. The purpose of this delay is to allow the processing results in that particular time interval to become stable. In particular, this is true for the immediate subscriptions for the SLSD (delayed by 1,200 seconds in the delivered configuration) and all subscriptions based on the automatic SELn origins (delayed by 2,400 seconds in the delivered configuration). Whereas the subscriptions based on the REB and the segment archive and all fileproduct subscriptions are not being artificially delayed. Refer to [\[Bor00\]](#) for a detailed discussion of product release times. The following text provides instructions for changing the artificial delays of subscription product releases.

Changing the SLSD Delay

The SQL script *Sta_into_DataReady.sql* is called from the parameter *stop-hook* in `$(CMS_CONFIG)/app_config/distributed/tuxshell/detpro/tuxshell-StaPro.par`. The delay of 1,200 seconds is passed as a literal argument from the *stop-hook* command line to the SQL script:

```
stop-hook="sqlwrap EXPERTDB \
@$(CMS_CONFIG)/system_specs/sql/Sta_into_DataReady.sql INITIAL \
%name %time %endtime 1200 $(timezone-difference)"
```

To change the delay:

1. Substitute the desired delay in seconds for the number 1,200 in the `stop-hook` command line.

Note: BE SURE THE RESULTS OF STATION PROCESSING CAN BE GUARANTEED TO BE STABLE BEFORE THE SLSD IS DELIVERED (TAKING INTO ACCOUNT OVERLAP IN *DFX* PROCESSING INTERVALS). THIS IS THE MILESTONE VALUE $T_{LOADDATE}$ (SEE TABLE 1 IN [\[BOR00\]](#)) FOR THE SUBSEQUENT STATION PROCESSING INTERVAL.

Changing SELn Product Delays

The SQL script *Origin_into_DataReady.sql* is called from the parameter *stop-hook* in the following par files:

```
$(CMS_CONFIG)/app_config/distributed/tuxshell/...
...sel1/tuxshell-WE-sel1.par
...sel2/tuxshell-WE-sel2.par
...sel3/tuxshell-Beamer.par
```

The delays are passed as symbolic parameters *WE-lookback* (for SEL1 and SEL2) and *Beamer-lookback* (for SEL3) from the `stop-hook` command line to the SQL script. As an example, see the *stop-hook* for SEL1:

```
stop-hook="sqlwrap EXPERTDB @$ (SQLDIR)/Origin_into_DataReady.sql \
sel1 %time %endtime $(WE-lookback) $(timezone-difference)"
```

The processes *WaveExpert* (WE) and *Beamer* use the same lookback values in their command lines and thus share the delay specified for the `stop-hook`. The parameters *WE-lookback* and *Beamer-lookback* are specified in the parameter file `$(CMS_CONFIG)/system_specs/shared.par`. In the standard configuration they are both set to 2,400 seconds.

To change the delays:

1. Change the parameter setting in `shared.par`. If the delays for SEL1 and SEL2 are desired to be different, define two separate parameters, for example, *WE-SEL1-lookback* and *WE-SEL2-lookback* in `shared.par` and reference them in the respective *tuxshell* par files.

▼ Operational Procedures

When choosing new delays, be sure to select multiples of 20 minutes for network processing intervals.

Note: BE SURE THE RESULTS OF NETWORK PROCESSING CAN BE GUARANTEED TO BE STABLE BEFORE THE ORIGIN-BASED SUBSCRIPTIONS GO OUT. SEE INFORMATION ON THE MILESTONE VALUE T_{STABLE} IN TABLE 1 OF [BOR00].

Adding New Fileproducts

The Subscription Subsystem is designed to support user-defined fileproducts. The software design that underlies this capability is illustrated in the following procedure, which is used to define a fileproduct.

To define a fileproduct:

1. Add a record describing the fileproduct to the **fpdescription** database table. The table is described in [IDC5.1.1Rev2].
2. Specify the required contents: a unique identifier (*typeid*), the product type, a description of the product, the type of data, and the format of the data.
3. After the description is created, generate a **fileproduct** row referencing the filesystem object.

The following example creates the **fpdescription** row (for station status rows):

```
SQL> describe fpdescription;
```

Name	Null?	Type
-----	-----	-----
TYPEID	NOT NULL	NUMBER(8)
PRODTYPE		VARCHAR2(12)
NAME		VARCHAR2(64)
MSGDTYPE		VARCHAR2(16)
MSGDFORMAT		VARCHAR2(8)
HEADER_FPID		NUMBER(8)
LDDATE		DATE

```
SQL> insert into fpdescription
  2 values (28,'STA_STATUS', 'station status reports',
  3 'ascii', 'IMS1.0', -101, sysdate);
1 row created
```

4. With the **fpdescription** row defined, configure the Subscription Subsystem to see the fileproduct. Modify the two parameter files: `ParseSubs.par` and `SubsProcess.par`. In `ParseSubs.par` modify the parameters: *supported_commands* and possibly *commands-station_constraint* (these are described in the *ParseSubs* man page). In both cases, add the product type (as defined in **fpdescription.prodtype**) to the parameter listing as follows:

```
supported_commands='WAVEFORM BULLETIN ... STA_STATUS ... RMSSOH'
commands-station_constraint='WAVEFORM ... STA_STATUS ... GASBKPHD'
```

Only add rows to *commands-station_constraint* if they can and should be constrained by the station name. By adding to this parameter, *ParseSubs* knows to generate a **producttypesta** row to allow for station-constrained subscriptions.

`SubsProcess.par` also requires a similar modification to the *supported_commands* parameter. In addition, set values for *command-[prodtype]*. In this case, with *prodtype* = `STA_STATUS`, the value is:

```
command-STA_STATUS=FILE_PRODUCT
```

Finally, the parameter *gselists* needs to include `FPID_LIST`, and a parameter *tagid-FPID_LIST* = `fpid` must be set. These last two are in the default configuration and should not require modification.

With fileproducts, the insert into the **dataready** table is automatic; the database trigger *fileproduct_to_dataready* executes on insert into the **fileproduct** table. [Table 2](#) shows the relationship between the **fileproduct** table and the **dataready** table:

▼ Operational Procedures

TABLE 2: FILEPRODUCT TO DATAREADY MAPPING

dataready Attribute	Contents
<i>dataid</i>	<i>lastid.keyvalue</i>
<i>tablename</i>	'fileproduct'
<i>account</i>	<i>fpdescription.prodtype</i>
<i>tagname</i>	'fpid'
<i>tagid</i>	<i>fileproduct.fpid</i>
<i>tagid2</i>	-1
<i>data_info</i>	<i>fileproduct.sta</i>
<i>status</i>	'n'
<i>lddate</i>	<i>fileproduct.lddate</i>

After the row is inserted into the **dataready** table, the database trigger *DataReady_to_Prodtrack* compares the value of **productcriteria.prodtype** to **dataready.account** and also the value of **producttypesta.sta** to the contents of **dataready.data_info**. If there is a match (for a particular *prodid*), then a row is added to the **prodtrack** table, and *SubsProcess* processes the row.

Thus, the only remaining step is to create the **fileproduct** row. For that, the program *write_fp* is called by the process that generates the new product encapsulated as a fileproduct (for example, *run_stacap* for STA_STATUS reports).

Configuring Signing of Subscriptions

ParseSubs is capable of signing the messages sent back to users as a confirmation of processing of the subscription request messages, and *SubsProcess* can ensure that outgoing subscriptions are signed. To accomplish this the parameter *support_smime* = 1 must be set and the following parameters must be set to suitable values: *ca_cert_path*, *sign_cert_path*, *sign_key_path*, *pass_phrase*. Sections

4.3.3–4.3.5 of [\[Oan01\]](#) explain the significance of these parameters. In the delivered standard configuration they are all set in `$(CMS_CONFIG)/system_specs/mss_auth.par`, which is read by all components of the Message and Subscription Subsystems.

[“Authentication” on page 62](#) explains the procedure for using a different signature on subscriptions than on other outgoing messages.

MAINTENANCE

The Subscription Subsystem requires no regular maintenance tasks with the exception of an occasional cleaning up of obsolete product descriptions (depending on your site’s policy in this respect).

Purging Log Files

Log files of the applications *SubsProcess* and *ParseSubs* are written to `MSGLOGDIR`, which is set in `$(CMS_CONFIG)/system_specs/msgs.par`, for example, to `$(LOGDIR)/msg`, which resolves to `/logs/msg`. These log files are regularly overwritten when their number reaches the configured maximum (20 in the delivered configuration), so they do not need to be purged manually. However, if log files are to be archived they must be copied before they are automatically purged. The purge does not occur at a pre-determined time, rather, they roll over when a parameter-specific number of log files have been written (usually 20).

Cleaning Up Obsolete Product Descriptions

To use computing resources economically, the Subscription Subsystem sets up a specific product definition only once, when the first user subscribes to the product. Later, when additional users subscribe to the same product, no new product definition is set up. Accordingly, when a user unsubscribes from a product, the product definition remains intact so that other users may continue to receive the product. As a consequence, a product definition becomes “orphaned” when the last user

▼ Operational Procedures

unsubscribes from it or when that subscription expires or is removed. This is true as well for individually constrained products, which presumably never attract more than one subscriber.

Therefore, over time, the Subscription Subsystem is expected to accumulate product definitions for which a subscriber no longer exists. These form a record of past system states and may be useful to “transfer” subscriptions to new users or as models to manually set up product definitions. In the end you may want to delete obsolete product descriptions, for example once a year.

The queries in this section identify all of the products for which no user is actively subscribed.

The following SQL query finds such obsolete product descriptions:

```
SQL> select pc.prodid, count(s.prodid)
2   from idcx.productcriteria pc, idcx.subs s
3   where 1=1 and s.prodid(+) = pc.prodid and s.status(+)='a'
4         and s.prodid is NULL
5   group by pc.prodid;
```

The following query deletes obsolete product descriptions using the above query as a subquery:

```
SQL> delete from producttypeorigin where prodid in
2   (select pc.prodid from idcx.productcriteria pc, idcx.subs s
3   where 1=1 and s.prodid(+) = pc.prodid and s.status(+)='a'
4         and s.prodid is NULL);
SQL> commit;
```

A similar delete query must be used for **productypesta** and **productypeevsc**.

SECURITY

Security for the Subscription Subsystem is provided by ownership of the processes. Operators can start and terminate the Subscription Subsystem if they are working with the same UNIX user login that has ownership of the processes.

If operators have update permissions for the database accounts they can potentially remove, add, or manipulate data in the account. For example, subscriptions for a particular nation or institution can be blocked or altered. Tables can be dropped, deleted, or truncated. Records from the tables can be removed or updated.

Authorized external users of the Subscription Subsystem are tracked by the **subsuser** table. Each user is identified by a unique user name and domain, which must match all email headers of messages sent to the subsystem to be processed. Discrimination between authorized and unauthorized users relies on their email headers being trustworthy, unless message signing and authentication is enabled.

Passwords

Passwords are not written to Subscription Subsystem log files.

Passwords to database accounts may be disclosed if a Subscription Subsystem program exits and generates a segmentation fault or core dump. To prevent this output a parameter in `global.env` limits the size of the core file generated.

Marking/Storing Controlled Outputs

All incoming subscription request messages and outgoing product messages are stored as files in the directory structure. Permissions on these files allow all operators and staff at the IDC to access these messages.

Because email is used as the primary method of transmitting messages, content of the messages is not secure during routing from the originator to the recipient. Email can be intercepted during transmission.

Data compiled per request and made available by FTP is stored as a file in a directory structure. An email notifies the requestor that the data are available for retrieval via FTP. Data files stored in this directory for retrieval are accessible by all users.

Chapter 3: Troubleshooting

This chapter describes how to identify and correct problems related to the Subscription Subsystem and includes the following topics:

- [Monitoring](#)
- [Interpreting Error Messages](#)
- [Solving Common Problems](#)
- [Error Recovery](#)
- [Reporting Problems](#)

Graphical User Interface (GUI). Within minutes of REB completion, you should receive the just-completed REB as a message in the mailbox for the specified return address. You can similarly monitor any other subscription type.

If a subscription is not received, follow the procedures described in the following sections to monitor the database tables and log files. In addition, refer to the section [“Solving Common Problems” on page 53](#), which contains a subsection on what to do if subscriptions are not received.

Monitoring Database Tables

You can monitor certain key database tables to determine if the Subscription Subsystem is operating normally. In particular, you can check product status, product queueing, and monitor the addition of rows to the **dataready** table.

Checking Product Statuses

To ensure that products are being processed after they are queued, use the following database query:

```
SQL> select prodid, status, count(*) from IDCX.PRODTRACK
      2 group by prodid, status;
```

Expect output similar to the following:

PRODID	STATUS	COUNT(*)
-----	-----	-----
1	DONE	2617
2	DONE	2405
4	DONE	9
5	DONE	46
6	DONE	45
8	NOACTIVE	2548
9	NOACTIVE	46
10	NOACTIVE	9
11	NOACTIVE	3
12	NOACTIVE	2
14	NOACTIVE	1

▼ Troubleshooting

15	NOACTIVE	1
25	DONE	2568
26	DONE	49

That is, for each product, the *status* should be either DONE or NOACTIVE. Any other *status* (NEW, FAILED, QUEUED, RUNNING) indicates that either there is a possible problem processing the subscription or the subscription is actively being processed. If you find any status of NEW, QUEUED, or RUNNING, wait a few minutes and rerun the query, as it is possible that products are being processed while you are checking the *status* values.

Check Product Queuing

You can use SQL statements to verify that products are being queued. As an example, the query below checks the latest entry in the queue for each product. It is primarily based on the **prodtrack** table, but joins with the **productcriteria** table to make items easier to read:

```
SQL> Select pc.prodtype, pc.prodsubtype, pc.prodid,
2  to_char(max(pt.lddate), 'yyyy/mm/dd hh24:mi') maxdate
3  from idcx.productcriteria pc, idcx.Prodtrack pt
4  where pc.prodid=pt.prodid
5  group by pc.prodtype, pc.prodsubtype, pc.prodid
6  order by pc.prodid;
```

The results of the query indicate any products that have not been delivered in a while. If so, further investigation is warranted.

PRODTYPE	PRODSUBTYP	PRODID	MAXDATE
BULLETIN	sel1	1	2001/07/04 13:01
BULLETIN	sel2	2	2001/07/04 12:50
BULLETIN	reb	4	2001/06/29 21:30
BULLETIN	sel1	5	2001/07/04 03:01
BULLETIN	sel2	6	2001/07/04 07:10
BULLETIN	sel1	8	2001/07/04 13:01
BULLETIN	sel1	9	2001/07/04 03:01
BULLETIN	reb	10	2001/06/29 21:30
BULLETIN	sel1	11	2001/07/04 10:21
BULLETIN	sel2	12	2001/07/03 20:51
BULLETIN	sel1	14	2001/07/04 03:01

BULLETIN	sel2	15	2001/07/04 07:10
STA_STATUS	station	25	2001/07/04 07:20
CHAN_STATUS	channel	26	2001/07/04 07:20
BULLETIN	seb	27	2001/06/29 22:20

In the example, the query was run at 2001/07/04 13:17 Greenwich Mean Time (GMT). At that time, most of the products were current. The dates for the REB-based products (REB and SEB) indicate that they are several days old; however, *rebdone* was last run at that time.

If you notice any products that should have been delivered but have not (for example, if an instant SEL subscription [*prodid*=1] had not been updated for several hours or days) there might be another problem. The following sections discuss how to monitor and diagnose such problems.

Monitoring Addition of Rows to *dataready* Table

To verify that database records are being added to the *dataready* table, use the following database query:

```
SQL> Select tablename, account, tagname,
2   to_char(max(lddate), 'yyyy/mm/dd hh24:mi') maxdate
3   from Idcx.dataready
4   group by tablename, account, tagname;
```

Sample results (for a correct setup) follow:

TABLERNAME	ACCOUNT	TAGNAME	MAXDATE
-----	-----	-----	-----
arrival	INITIAL	jdate	2001/07/04 07:40
arrival	INITIAL	time	2001/07/04 13:00
arrival	INITIAL	time-inst	2001/07/04 13:00
evchar	idcreb	jdate	2001/06/29 21:20
fileproduct	ARR	fpid	2001/07/04 05:47
fileproduct	BLANKPHD	fpid	2001/06/01 22:03
fileproduct	CHAN_STATUS	fpid	2001/07/04 06:20
fileproduct	DETBKPHD	fpid	2001/05/24 23:49
fileproduct	MET	fpid	2001/07/04 12:44
fileproduct	QCPHD	fpid	2001/07/04 05:38
fileproduct	RMSSOH	fpid	2001/07/04 12:44
fileproduct	RNPS	fpid	2001/07/04 05:33

▼ Troubleshooting

fileproduct	RRR	fpid	2001/07/02 16:06
fileproduct	SPHDF	fpid	2001/07/04 05:47
fileproduct	SPHDP	fpid	2001/07/04 12:24
fileproduct	SSREB	fpid	2001/07/02 13:50
fileproduct	STA_STATUS	fpid	2001/07/04 06:20
fileproduct	tm_detection	fpid	2001/07/04 12:42
fileproduct	tm_status	fpid	2001/07/04 12:41
fileproduct	tm_uptime	fpid	2001/07/04 12:42
origin	reb	jdate	2001/06/29 20:30
origin	sel1	jdate	2001/07/04 02:01
origin	sel1	time	2001/07/04 13:00
origin	sel1	time-inst	2001/07/04 13:00
origin	sel2	jdate	2001/07/04 06:10
origin	sel2	time	2001/07/04 12:10
origin	sel2	time-inst	2001/07/04 12:51
wfdisc		wfid	2001/07/04 13:01
wftag	segment	orid	2001/06/29 21:00

If something is clearly old, then investigate further. If the **dataready.tablename** is **fileproduct**, check to see if any records for that data type have been added to the **fileproduct** table. If so, then the database trigger **fileproduct_to_dataready** may be disabled. To check the status of the trigger, use the following SQL:

```
SQL> select trigger_name, trigger_type, triggering_event, status
2   from user_triggers
3  where trigger_name='FILEPRODUCT_TO_DATAREADY';
```

Expect the following results:

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	STATUS
FILEPRODUCT_TO_DATAREADY	AFTER EACH ROW	INSERT	ENABLED

If this is not the case, reinstall the trigger (see [“Database Triggers” on page 73](#)).

If the **dataready.tablename** is not **fileproduct**, the operator should check the log files of the process inserting the **dataready** row (see [Check Log Files](#) below).

Check Log Files

For non-fileproduct based data regularly (for example, daily) check the appropriate log files to identify possible problems.

Log files for *ParseSubs* and *SubsProcess* are written to MSGLOGDIR (for example, /logs/msg). A specified number (parameter log-files, currently set to 20 for both applications) of log files is written. The most current log files are *ParseSubs* and *SubsProcess*; files written immediately prior to the current files are *ParseSubs.1* and *SubsProcess.1*; and the files before that are *ParseSubs.2* and *SubsProcess.2*, and so on, up to *ParseSubs.20* and *SubsProcess.20*. The applications themselves purge the old log files. For example:

```
$ cd /logs/msgidc012:/logs/msg
$ ls -lart ParseSubs*

-rw-r--r--  1 auto      auto 252 Jun 15 09:39 ParseSubs.20
-rw-r--r--  1 auto      auto 162 Jun 15 09:42 ParseSubs.19
...
-rw-r--r--  1 auto      auto 162 Jul  2 11:07 ParseSubs.2
-rw-r--r--  1 auto      auto 162 Jul 10 07:47 ParseSubs.1
-rw-r--r--  1 auto      auto 162 Jul 10 11:59 ParseSubs

$ ls -lart SubsProcess*

-rw-rw-r--  1 auto      auto 74133 Aug  5 2000 SubsProcess.20
-rw-r--r--  1 auto      auto 80792 Aug  9 2000 SubsProcess.19
...
-rw-r--r--  1 auto      auto 152669 May 18 09:44 SubsProcess.2
-rw-rw-r--  1 auto      auto 447358 Jun  5 04:50 SubsProcess.1
-rw-r--r--  1 auto      auto 957429 Jul 10 14:41 SubsProcess
```

Unusual file sizes or particularly rapid turnover of log files can be an indication of problems. The touch dates of such unusual log files give an idea of when the problem may have originally occurred. The “touch” date is always the time of the last entry into the log file. *ParseSubs* starts a new log file for each subscription request, the touch dates of its log files indicate approximately when the last 20 subscription request messages were processed; however, *SubsProcess* appends to the currently open log file until a maximum size is reached (unless it is shut down and restarted, in which case a new log file is started even if the old one was not “full” yet). In the previous example, *SubsProcess.1* spans the time period between May 18 09:44 and Jun 5 04:50.

▼ Troubleshooting

To scan expeditiously for error messages, *grep* for the words *fatal*, *error*, or *warning* from the log files:

```
$ grep fatal ParseSubs* SubsProcess*
$ grep error ParseSubs* SubsProcess*
$ grep warning ParseSubs* SubsProcess*
```

If any rows are returned, consult the next section “[Interpreting Error Messages](#)”.

The log files written by *AutoDRM* (a component of the Message Subsystem), when operating as a child process of *SubsProcess* to assemble and ship subscribed products, are also relevant. They are written to the same MSGLOGDIR as the log files of *ParseSubs* and *SubsProcess*. The most recent log file is named *AutoDRM_subs*, and the older ones *AutoDRM_subs.n* in the fashion described above.

Finally, processes that make products available for distribution via the Subscription Subsystem write log files in different locations, as indicated in [Table 3](#).

TABLE 3: LOGS AND DATAREADY ROWS

DB Table	Subtype	Products	Log File
arrival	initial	arrival:automatic	LOGDIR/ <i>jdate</i> /tuxshell/StaPro*
evchar	idcreb	SEB, SSEB, NEB, NSEB	LOGDIR/ <i>jdate</i> /Evch/evsc*
origin	sel1	arrival:associated, bulletin, event, origin for SEL1	LOGDIR/ <i>jdate</i> /tuxshell/WE-sel1*
origin	sel2	arrival:associated, bulletin, event, origin for SEL2	LOGDIR/ <i>jdate</i> /tuxshell/WE-sel2*
origin	sel3	arrival:associated, bulletin, event, origin for SEL3	LOGDIR/ <i>jdate</i> /tuxshell/beamer*

TABLE 3: LOGS AND DATAREADY ROWS (CONTINUED)

DB Table	Subtype	Products	Log File
origin	reb	arrival:associated, bulletin, event, origin for REB	LOGDIR/ <i>jdate</i> /tuxshell/ol2or*
wftag	segment	waveforms from events	LOGDIR/ <i>jdate</i> /tuxshell/doday*
wfdisc		auxiliary waveforms	LOGDIR/msgs/ParseData*

In the *tuxshell* log files look for stop-hook execution. In the event screening (*evsc*) log files look for the notice "insert into idcx.dataready." In the *ParseData* log files look for messages like "Can't insert dataready into database."

INTERPRETING ERROR MESSAGES

ParseSubs Error Messages

The following errors may be encountered from the *ParseSubs* process:

Message: [error]:type=parse, text=Unknown command TIME_STAMP
[error]:type=parse, text=Unknown command ARMR

Description: *ParseSubs* encountered the TIME_STAMP command, which does not work with subscriptions.

Action: Inform the user that TIME_STAMP does not work with subscription. Check to verify that the ARMR request is formatted correctly. If so, verify that ARMR is in the supported commands parameter.

▼ Troubleshooting

Message: [error]: type=internal, text=Error: could not delete subscription

Description: The user attempted to delete a subscription, but the action was unsuccessful.

Action: Find the message based on the *msgid*. Ensure that the user was attempting to unsubscribe from his/her own subscriptions and that the message was formatted properly. If so, unsubscribe the subscription manually by updating the *status* field for the **subs** table to 'i', send an email to the user informing him/her that the subscription has been terminated, and investigate why the subscription could not be deleted.

Message: [error]: type=internal, text=Can't insert producttypeevcsc rows

Description: *ParseSubs* was unable to add the rows needed for the new product.

Action: Ensure that the tables exist and are writable. If so, then try rerunning the subscription request by updating the **msgdisc.status** (update **msgdisc** set **status='RECEIVED'** where **msgid=number**). If it fails again, it is probably a software defect. See ["Reporting Problems" on page 58](#).

SubsProcess Error Messages

The following errors may be observed from *SubsProcess*:

Message: [error]: Unable to get subs, query = ...
 [error]: Unable to get producttypes row, query = ...
 [error]: Unable to get productcriterias row, query = ...
 [error]: Unable to get data, query = ...
 [severe]: unable to get product criteria, query= ...

Description: An error was encountered while trying to select from **subs**, **productcriteria**, **producttypesta**, **producttypeorigin**, or **dataready**.

Action: Check the (printed) query by cutting and pasting into an SQL*Plus window. If the query does not report an error (0 rows is okay), then kill and restart *SubsProcess*; otherwise, figure out why the query is failing. It is probably a database issue.

Message: [error]: There is no dataready row with dataid=[]

Description: For some reason the **dataready** row does not exist.

Action: Check the **dataready** table for *dataid*. If it exists, then try running *SubsProcess* again. If it does not exist then: (1) confirm that someone actually deleted the **dataready** row or (2) someone manually entered a **prodtrack** row. Then, delete the lone **prodtrack** row.

▼ Troubleshooting

Message: [warning]: Process exited with status []
followed by:
[warning]: Product id=[], delivery id=[] not delivered.

Description: The child process (*AutoDRM*, a component of the Message Subsystem) encountered an error.

Action: First check the *AutoDRM_subs* log file (in the same directory as the *SubsProcess* log files). Identify the problem, fix the problem, and then requeue the product delivery (update `prodtrack.status` set `status='NEW'` where `prodid=[]` and `delivid=[]`).

Message: [error]: the value of product.status was not [] for
`prodid=[]`, `dataid=`
[severe]: unable to queue status, query = ...

Description: The value of the status attribute was not what was expected.

Action: Probably more than one instance of *SubsProcess* is running. If so kill off all active *SubsProcess* instances and restart. It is possible that the other *SubsProcess* is running on a different machine.

Message: [fatal]: Could not open database [], vendor []

Description: *SubsProcess* was unable to open the database connection.

Action: Ensure that the database is running; check environments `GDIHOME` and `ORACLE_HOME`; check that the database passwords have not changed.

Message:	[error]: unable to read subscription queue [error]: unable to process subscription queue
Description:	Reading or processing the queue failed; the message has been preceded by another error message describing the problem.
Action:	Fix the earlier problem, and then requeue and rerun the message.

Message:	[fatal]: AutoDRM Killed
Description:	Reaction to a SIGTERM, for example, via UNIX <i>kill</i> command.
Action:	No action if this occurred as part of a regular shutdown.

Message:	[fatal]: SEGMENTATION FAULT, Core Dumped
Description:	A software defect was encountered.
Action:	See "Reporting Problems" on page 58 .

SOLVING COMMON PROBLEMS

Subscription Request Fails

Subscription requests often fail with first-time Subscription Subsystem users. To diagnose and solve this problem:

1. Check if the user is not in the **subsuser** table.
Follow the instruction in ["Adding New Users" on page 22](#).
2. Check if the user committed syntax errors in the Subscription Request Message.
Find the syntax error, formulate a well-formed Subscription Request Message, and send it to the user. If problems persist, consider setting up the subscriber's first subscription manually.

▼ Troubleshooting

Unclear Message from SubsProcess to Subsuser

A subscriber may receive an unclear message, for example “db error,” from *SubsProcess*. This may be the result of a syntax or semantics error in the subscription request, coupled with unhelpful diagnostics. To solve this problem:

1. Find and troubleshoot the Subscription Request Message that caused the error.

If you identify a syntax error, formulate a well-formed Subscription Request Message and send it to the user. If you identify a semantics error (formally correct request that makes no sense) explain this to the user.

Product Delivery Fails

Another common problem is a product delivery failure (no products go out). To solve this problem:

1. Check if *SubsProcess* is not running.
Restart *SubsProcess*.
2. Check if *SubsProcess* is “wedged” (in process table but inactive).
Kill and restart *SubsProcess*.
3. Check if outgoing email is blocked.
Troubleshoot the mail server, and request help from the infrastructure staff.
4. Check if there is a firewall problem.
Troubleshoot the firewall configuration, and request help from the infrastructure staff.

One User Stops Receiving One Product

A single user may stop receiving a specific product. To solve this problem:

1. Check if the subscription has expired (`subs.offdate < today`).
Do not update `subs.offdate`, but create a new `subs` row like the old one, with `ondate = today` and `offdate > today`.

One User Stops Receiving All Products

A single user may stop receiving all products. To solve this problem:

1. Check the `subsuser.status` to see if the `status` has been changed to "i" (inactive).
2. Check if the user's email address is misspelled or recently changed.
Verify the address with a manual test email. Update `subsuser.username`, `subsuser.domain` if required.
3. Check if the user's daily quota has been exceeded.
Inform the user accordingly.

All Users Stop Receiving One Product

All users may stop receiving one product. To solve this problem:

1. Check if the product has not been created.
Find and correct the problem in automatic processing.
2. Check if the product is not registered by the Subscription Subsystem.
Follow the instructions in ["Monitoring" on page 42](#):
 - Check product status.
 - Check product queueing.
 - Check addition of rows to `dataready`.

One or More Users Do Not Receive a Product

One or more users may not receive a product that was sent. To solve this problem:

▼ Troubleshooting

1. Use the following SQL*Plus query to find the *msgid* for the product that was not delivered:

```
SQL> select msgid from prodtrack  
      2 where prodid=product_id and delivid=delivery_id;
```

2. Update the **msgdest** row to **PENDING** for the users that did not receive the product:

```
SQL> update msgdest set status='PENDING'  
      2 where msgid=message_id and mailto=email_address;  
SQL> commit;
```

Empty Fileproduct Delivered

Occasionally users receive messages with only headers and no information. To diagnose and correct this problem:

1. Check if Automatic Processing has generated empty flat files.
Find and correct the problem in Automatic Processing.

Subscription is Incomplete/Defective

Occasionally users receive messages that are not what is expected. To diagnose and correct this problem:

1. Check if the product itself is incomplete/defective.
Find and correct the problem in Automatic Processing.
2. Check if the subscribed data are not available on the operational system (for example, station on testbed).
Explain this to the user.
3. Check if the user constrained a subscription in a way that caused unexpected results.
Troubleshoot the subscription, and suggest a version that achieves the user's objective.

4. If you suspect a software defect in the Subscription Subsystem, follow the instructions in [“Reporting Problems” on page 58](#).

Duplicated Products Delivered

Occasionally users receive duplicate messages. To diagnose and correct this problem:

1. If one user received duplicate subscriptions, check that the user does not have multiple subscriptions for the same product by using the following database query:

```
SQL> select prodid, count(*) from subs  
2 where userid=id group by prodid;
```

2. If all users received duplicate subscriptions, check if a duplicate or cloned entry exists in the **dataready** table (for example, due to REB being migrated twice or because an interval was reprocessed). Send a broadcast message to the subsusers and ask them to disregard either the first or the second delivery.

ERROR RECOVERY

If *SubsProcess* crashes, wedges, or is killed while processing subscriptions, these subscriptions may remain stuck in a status that is supposed to be transient. Use the following procedure to recover in this case:

1. If *SubsProcess* is still in the process table, shut it down completely, if necessary, using `$ kill -9 <pid for SubsProcess>`.
2. Check Product Statuses using the procedure provided in [“Monitoring Database Tables” on page 43](#).
3. Update **prodtrack.status** to **NEW**, where (a) the current status is transitory (**QUEUED**, **RUNNING**), (b) the current status is **FAILED**, and (c) the current status is **DONE**, but the subscription has not been sent out.
4. Restart *SubsProcess* (following the procedure given in [“Software Startup” on page 16](#)).

▼ Troubleshooting

In other cases, updating `prodtrack.status` to `NEW` also will give the associated product another chance to be distributed by the Subscription Subsystem.

REPORTING PROBLEMS

The following procedures are recommended for reporting problems with the application software:

1. Diagnose the problem as far as possible.
2. Record information regarding symptoms and conditions at the time of the software failure.
3. Retain copies of relevant sections of application log files.
4. Contact the provider or maintainer of the software for problem resolution if local changes of the environment or configuration are not sufficient.

Chapter 4: Installation Procedures

This chapter provides instructions for installing the software and includes the following topics:

- [Preparation](#)
- [Executable Files](#)
- [Configuration Data Files](#)
- [Text Files](#)
- [Database](#)
- [Initiating Operations](#)
- [Validating Installation](#)

Chapter 4: Installation Procedures

PREPARATION

Obtaining Released Software

The software is obtained via FTP from a remote site or via a physical medium, such as tape or CD-ROM. The software and associated configuration data files are stored as one or more tar files. The software and data files are first transferred via FTP or copied from the physical medium to an appropriate location on a local hard disk. The tar files are then untarred into a standard UNIX directory structure and imported into a Configuration Management System if desired.

Hardware Mapping

The user must select the hardware on which to run the software components. Software components are generally mapped to hardware to be roughly consistent with the software configuration model. It is recommended for the Subscription Subsystem to be co-configured on the machine selected for the Message Subsystem.

Establishing Target Information

To follow the instructions below it is useful to establish certain target information first. In particular:

- Establish which machine is running the Message System (MSGHOST, for example, kuredu at the IDC)

- Establish the directory where executable binaries are installed (RELBIN, for example, /cmss/rel/bin).
- Establish the directory where executable scripts are installed (SCRIPTSBIN, for example, /cmss/scripts/bin).
- Establish the path to the main tree of configuration data files (CMS_CONFIG, for example, /cmss/config).
- Establish the path to and the name of the CSCI-level parameter file for the Message Subsystem (MESSAGES, for example, /cmss/config/system_specs/msgs.par).
- Establish the path to the CSC-level parameter file directories for Message Subsystem and Subscription Subsystem (MESSAGES-DIR, for example, /cmss/config/app_config/messages).
- Establish the user name under which the Message System is run (AUTO, for example, auto).
- Establish the symbolic form of the connect string needed to connect to the database account on which the Message System operates (for example, IDCXDB).
- Establish the directory under which SQL code resides on the system (SQLDIR, for example, /cmss/rel/sql).
- Establish the directory that is used by the Message Subsystem and also by the Subscription Subsystem to write log files (MSGLOGDIR, for example, /logs/msg).
- Establish the email address to which subscription request messages are to be sent. This is the functional mail-alias used by the Message Subsystem for incoming messages (MSG_RETURN_ADDRESS, for example, messages@idc.org [\[IDC6.5.19\]](#)).

UNIX System

No preparation is needed beyond the above for the Message Subsystem (see Chapter 4 of [\[IDC6.5.19\]](#)).

▼ Installation Procedures

Authentication

Incoming subscription request messages are authenticated by the Message Subsystem, so no additional action is required.

In the standard configuration, the Subscription Subsystem signs outgoing messages to subscribers with the same key that is used by the Message Subsystem. A separate key can be used if desired by making the following parameter change:

1. Generate a separate key pair (`subscert.pem` and `subsreq.pem`) for signing subscription messages, and place it in a directory called `CERTDIR`.
2. When determining the target configuration, set the following parameters in `ParseSubs.par` and `AutoDRM_subs.par`, thus overriding the settings in `MESSAGES`, to see the new keys.

```
sign_cert_path=$(CERTDIR)/subscert.pem
```

```
sign_key_path=$(CERTDIR)/subsreq.pem
```

The CA-certificate will be the same one as for the Message Subsystem, so you do not need to change the `ca_cert_path`.

For details see sections 4.3.3–4.3.5 of CMR-01/06, [\[Oan01\]](#).

EXECUTABLE FILES

Executable files are listed in [Table 1 on page 9](#). If desired, build the executable binaries from the delivered source code.

To install the executable files:

1. Shut down the Subscription Subsystem following instructions given in [“Software Shutdown” on page 18](#).
2. Copy executable binaries into `RELBIN`.
3. Copy executable shell scripts into `SCRIPTSBIN`.
4. Restart the Subscription Subsystem following instructions given in [“Software Startup” on page 16](#).

CONFIGURATION DATA FILES

All configuration data files are located in the configuration tree below `CMS_CONFIG`. They are delivered in their respective places both in full releases and in patch releases. After importing the release, site-specific configuration information should be merged into the delivered files before the merged files are installed in the run-time system. The details of these actions depend on the type of configuration management (system and procedures) in use at the site.

The following sections identify all explicit configuration data relevant to the Subscription Subsystem. All existing parameters are discussed in the man pages for the individual module, but not all of these parameters are explicitly set with default values applying in the absence of explicit settings.

Global Parameter Files

Configuration data files follow a hierarchy and make use of global variables wherever possible. The objective of this hierarchy is: (1) to avoid having to store the same information in more than one place, and (2) to centralize site-specific information in one place, namely the directory `$(CMS_CONFIG)/system_specs`. Lower-level parameter files *source* higher-level parameter files. All Subscription Subsystem parameter files *source* the following files:

- `$(CMS_CONFIG)/system_specs/process.par`
- `$(CMS_CONFIG)/system_specs/shared.par`
- `$(MSS_AUTH_PAR)`, which resolves for example, to `mss_auth.par`
- `$(MESSAGES)`, which resolves for example, to `msgs.par`

Also, `$(CMS_CONFIG)/system_specs/env/global.env` contains the shell environment variables under which all modules are run. A listing of the portions of these global (and CSCI) parameter and environment files that are relevant to the Subscription Subsystem follows.

▼ Installation Procedures

global.env

```

CMS_CONFIG=/cmss/config
CMS_HOME=/cmss/rel
CMS_MODE=process
CMS_SCRIPTS=/cmss/scripts
GDIHOME=/cmss/rel
GDI_HOME=/cmss/rel

IMSPAR=/cmss/config/system_specs/process.par
LD_LIBRARY_PATH=/usr/dt/lib:/cmss/rel/lib:/home/oracle/lib:
    /cmss/cots/tuxedo/lib:/opt/SUNWspro/lib:/opt/SUNWmotif/lib:
    /usr/openwin/lib:/cmss/contrib/lib

LOCALHOME=/cmss/local

MANPATH=/usr/man:/usr/openwin/man:/usr/dt/man:/cmss/rel/doc/man:
    /cmss/local/man

ORACLE_HOME=/home/oracle
ORACLE_SID=oracle

PATH=/usr/bin:/usr/sbin:/usr/dt/bin:/cmss/scripts/bin:
    /cmss/rel/bin:/cmss/contrib/bin:/home/oracle/bin:
    /cmss/cots/tuxedo/bin:/cmss/local/bin:/opt/SUNWspro/bin:
    /opt/SUNWmotif/bin:/usr/openwin/bin:
    /home/licensed/frame_5.5/bin:/opt/atria/bin:/opt/local/bin:
    /opt/local/adobe/Acrobat3/bin:/etc:/usr/ccs/bin:/usr/ucb:.

PERLLIB=/cmss/scripts/lib
PERLPATH=/cmss/local/bin/
SPROHOME=/opt/SUNWspro

```

process.par (IMSPAR)

```

IDCXDB=account/password@machine
EXPERTDB=$(IDCXDB)
par=$(CMS_CONFIG)/system_specs/shared.par

```

shared.par

```

operator=name, name, name
DATABASE_VENDOR=oracle

```

```
# Shared Directories
LOGDIR=/logs
MSGDIR=/messages

# Shared Directories with derived paths
RELDIR=$(CMS_HOME)
RELBIN=$(RELDIR)/bin
SQLDIR=$(CMS_CONFIG)/system_specs/sql
GLOBALDIR=$(CMS_CONFIG)/system_specs
CONTRIB_BIN=$(CONTRIB_HOME)/bin
SCRIPTSBIN=$(CMS_SCRIPTS)/bin

# Each subsystem has a fixed path to it.
MESSAGES-DIR=$(CMS_CONFIG)/app_config/messages

# The following define the subsystem specific par files
MESSAGES=$(CMS_CONFIG)/system_specs/msgs.par

# machines
SUBHOST=TUXHOST3
TUXHOST3=machinename
```

mss_auth.par

```
#
# Authentication parameters section
#

# 1) common pars for verification and signing
as-verbose=1
cache-life=600

# 2) for verification and signing
#   ca-cert-apth can be either filename or dir
ca-cert-path=$(MSS_AUTH_DIR)/CAcerts

# 3) for verification
#smime-verify=1
smime-verify=0

# 4) for signing
# cert-cache-path and priv-key-path are dirs (not filenames)
smime-sign=1
auth-app-subject=MSS
cert-cache-path=$(MSS_AUTH_DIR)/certs
```

▼ Installation Procedures

```
# if cert index file name is not "index".txt, you can specify it
# through the parameter cert-index-filename
#cert-index-filename=

priv-key-path=$(MSS_AUTH_DIR)/private

# if private key index file name is not "index.txt", you can specify
# it through the parameter priv-index-filename
#priv-index-filename=

# set proper value here to retrieve private key
auth-pass-phrase=tb.phr
```

msgs.par

```
PARDIR=$(MESSAGES-DIR)

# Common log directory for all message system
# and subscription system applications
MSGLOGDIR=$(LOGDIR)/msg
```

CSC-level parameter files

Parameter files on the level of an individual module are found under `MESSAGES-DIR/module-name/`. The CSC-level parameter files are all explicitly mentioned in [Table 1 on page 9](#). They should not contain site-specific information. Consequently, no changes should be required to these files during an installation. Nevertheless, it is recommended to compare the newly delivered parameter files with their currently installed versions, (for example, using the UNIX command *diff* and account for any differences found).

ParseSubs.par

```
par=$(IMSPAR)
par=$(MESSAGES)
par=$(MSS_AUTH_PAR)

database=$(EXPERTDB)
vendor=oracle
msg-account=idcx
subs-account=idcx
msgdir=$(MSGDIR)
```

```

aux_affiliation=AUX

log-directory=$(MSGLOGDIR)
log-name=ParseSubs
log-files=20
log-drm
log-gdi

fatal-error-email='$(operator) '

ValidDataFormat="GSE2.0 GSE2.1 IMS1.0 IMS2.0 RMS1.0 RMS2.0"
DefaultFormat=GSE2.0
msgdest-status-pending=PENDING
magdiff=0.01
geogdiff=0.01
depdiff=0.01

user-notauthorized_file=$(PARDIR)/ParseSubs/NotAuthorized

supported_commands='WAVEFORM BULLETIN ORIGIN EVENT DETECTION \
OUTAGE STA_STATUS CHAN_STATUS SPHDF SPHDP BLANKPHD CALIBPHD QCPHD \
DETBKPHD ARR RRR SSREB RMSSOH RNPS MET ALERT_TEMP ALERT_UPS \
ALERT_FLOW ALERT_SYSTEM ARRIVAL \
ARRIVAL:AUTOMATIC ARRIVAL:ASSOCIATED RLR GASBKPHD'

commands-station_constraint='WAVEFORM BULLETIN ORIGIN EVENT \
DETECTION OUTAGE STA_STATUS CHAN_STATUS SPHDF SPHDP BLANKPHD \
CALIBPHD QCPHD DETBKPHD ARR RRR SSREB RMSSOH RNPS MET ALERT_TEMP \
ALERT_UPS ALERT_FLOW ALERT_SYSTEM \
ARRIVAL ARRIVAL:AUTOMATIC ARRIVAL:ASSOCIATED RLR GASBKPHD'

commands-origin_constraint='BULLETIN ORIGIN EVENT'
commands-bull_type='BULLETIN ARRIVAL ORIGIN EVENT'
commands-relative_to='WAVEFORM'

bull_type_list=" \
IDC_SEL1 sel1 IDC_SEL1\
IDC_SEL2 sel2 IDC_SEL2\
IDC_SEL3 sel3 IDC_SEL3\
IDC_SEB seb IDC_SEB\
IDC_SSEB sseb IDC_SSEB\
IDC_REB reb IDC_REB\

```

▼ Installation Procedures

```

SEL1 sel1 IDC_SEL1\
SEL2 sel2 IDC_SEL2\
SEL3 sel3 IDC_SEL3\
SEB seb IDC_SEB\
SSEB sseb IDC_SSEB\
REB reb IDC_REB\
IDCSEL1 sel1 IDC_SEL1\
IDCSEL2 sel2 IDC_SEL2\
IDCSEL3 sel3 IDC_SEL3\
IDCSEB seb IDC_SEB\
IDCSSEB sseb IDC_SSEB\
IDCREB reb IDC_REB\
neb* neb* neb*\
nseb* nseb* nseb*\
*neb *neb *neb\
*nseb *nseb *nseb"

neb_nseb-bulltypes="neb* nseb* *neb *nseb"
affiliation_table=AFFILIATION
auxiliary_value=CUR_AUX
primary_value=CUR_PRI

ARRIVAL:ASSOCIATED-req_envs='bull_type sta_list'
ARRIVAL:AUTOMATIC-req_envs='sta_list'
BULLETIN-req_envs='bull_type'
EVENT-req_envs='bull_type'
ORIGIN-req_envs='bull_type'

#Default granularity for REB subscriptions
determined_granularity='ARRIVAL:REB:24 BULLETIN:REB:24 \
ORIGIN:REB:24 EVENT:REB:24 '

```

TEXT FILES

The Subscription Subsystem makes use of one text file containing a message that is sent to non-authorized users informing them how they can become authorized users. The name and path to this file is set in `$(MESSAGES-DIR)/ParseSubs/ParseSubs.par`:

```
user-notauthorized_file=$(PARDIR)/ParseSubs/NotAuthorized
```

This file should be customized before startup.

DATABASE

This section describes database elements required for operation of this software component, including accounts, tables, triggers, and initialization of **lastid**.

Accounts

The Subscription Subsystem interacts with tables in the primary pipeline database account (for example, IDCX). Under normal circumstances this account already exists when an installation of the Subscription Subsystem is performed. Instructions for the configuration of this and the other IDC database accounts are provided in [\[IDC5.1.3Rev0.1\]](#). Formalized descriptions and scripts to create the tables and synonyms (including grants) are provided under `$(CMS_CONFIG)/system_specs/ops_db/idcx` in terms of the *dumptable/loadtable/describe* functionality (see respective man pages).

Tables

The database account for the Subscription Subsystem must contain the tables that are part of the Message Subsystem functional group ([\[IDC7.4.2\]](#) and [\[IDC6.5.19\]](#)) and the tables used by the Subscription Subsystem. Only these latter tables are listed in [Table 4](#). [Figure 3](#) shows the entity-relationships between the tables of the Subscription Subsystem.

TABLE 4: SUBSCRIPTION SUBSYSTEM DATABASE TABLES

Table	Description
dataready	This table contains information about the availability of each individual product.
fpdescription	This table describes product types available as flat files.
fileproduct	This table describes files holding data products.
prodtrack	This table links the product definitions to the product deliveries and joins them to the data tables. It also allows tracking of individual products to outgoing messages.

▼ Installation Procedures

TABLE 4: SUBSCRIPTION SUBSYSTEM DATABASE TABLES (CONTINUED)

Table	Description
productcriteria	This table contains definitions of products to which one or more users have subscribed.
producttype(*)	These tables (with * standing for sta , origin , evsc) define constraints for products based on station, origin and event screening criteria, respectively.
subs	This table contains information about individual subscriptions.
subsuser	This table contains information on authorized users of the Subscription Subsystem.

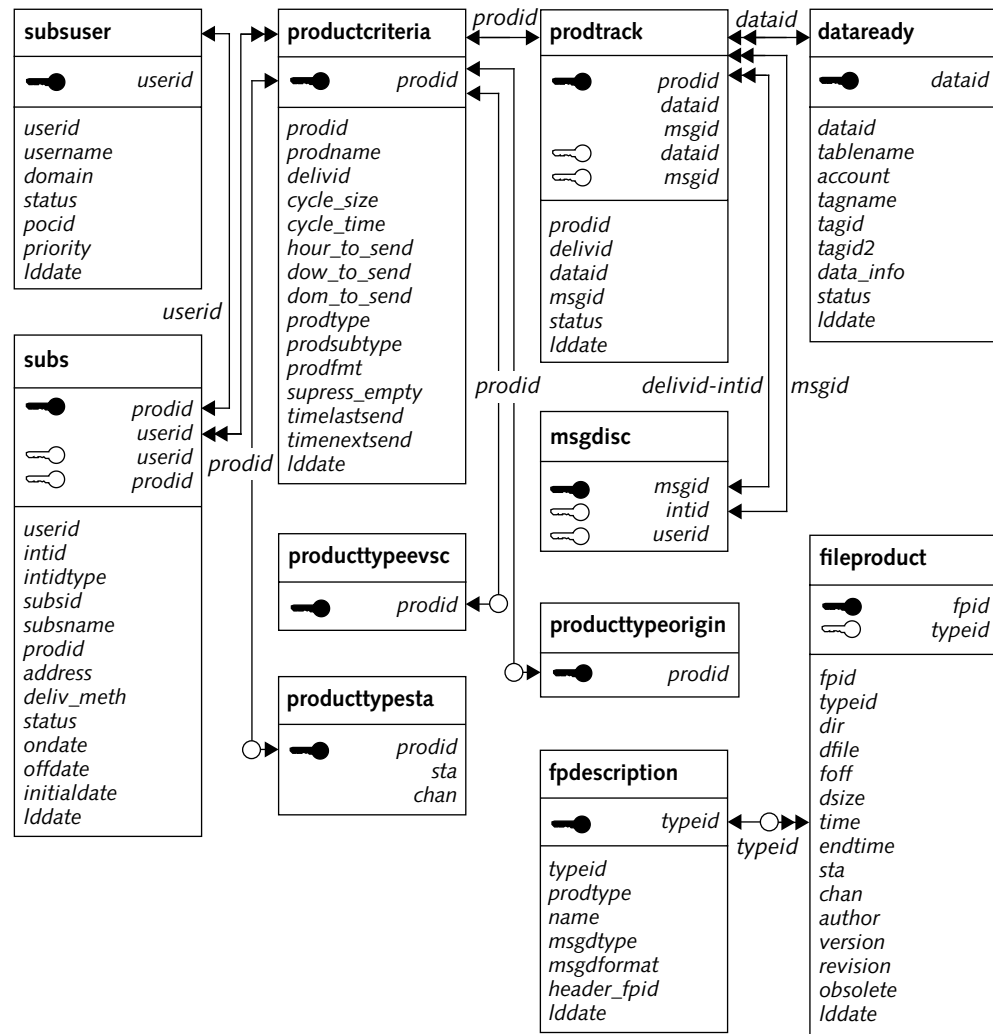


FIGURE 3. ENTITY RELATIONSHIPS OF SUBSCRIPTION SUBSYSTEM TABLES

Attributes, variable types and length, indexes, dimensions, and other information about Subscription Subsystem tables can be found in [\[IDC5.1.1Rev2\]](#) and [\[IDC5.1.3Rev0.1\]](#). Use the following command sequence to create all the tables and synonyms:

```
% cd CMS_CONFIG/system_specs/ops_db/idcx
```

▼ Installation Procedures

```
% sqlwrap IDCXDB @build_tables
% sqlwrap IDCXDB @build_syms
```

Build individual tables as follows:

```
% cd CMS_CONFIG/system_specs/ops_db/idx/sql
% sqlwrap IDCXDB @DATAREADY
% sqlwrap IDCXDB @FPDESCRIPTION
% sqlwrap IDCXDB @FILEPRODUCT
% sqlwrap IDCXDB @PRODTRACK
% sqlwrap IDCXDB @PRODUCTCRITERIA
% sqlwrap IDCXDB @PRODUCTTYPESTA
% sqlwrap IDCXDB @PRODUCTTYPEORIGIN
% sqlwrap IDCXDB @PRODUCTTYPEEVSC1
% sqlwrap IDCXDB @SUBS
% sqlwrap IDCXDB @SUBSUSER
```

You must drop existing tables before they can be (re)created.

The following example shows the SQL script *PRODUCTCRITERIA.sql*:

```
create table PRODUCTCRITERIA (
    prodid      NUMBER(8),
    prodname    VARCHAR2(24),
    delivid     NUMBER(8),
    cycle_size  NUMBER(8),
    cycle_time  NUMBER(4),
    hour_to_send NUMBER(2),
    dow_to_send NUMBER(2),
    dom_to_send NUMBER(2),
    prodtype    VARCHAR2(12),
    prodsubtype VARCHAR2(12),
    prodfmt     VARCHAR2(6),
    supresempty CHAR(2),
    timelastsend FLOAT(54),
    timenextsend FLOAT(54),
    lddate      DATE
    tablespace IDC storage (initial 100k next 100k
        pctincrease 0) pctfree 10;
create unique index PRODCRITX on PRODUCTCRITERIA(PRODID)
```

1. The script *producttypeevsc_cre.sql* was delivered in directory
/cmss/config/system_specs/ops_db/R3_updates.

```
tablespace IDCNDX storage (initial 100k next 100k pctincrease 0) p
ctfree 0;
```

```
grant SELECT on PRODUCTCRITERIA to PUBLIC with grant option;
```

Database Triggers

Two triggers are used by the Subscription Subsystem; their creation is described below. If a trigger of the same name already exists in the account, the existing trigger is updated rather than first dropped and then recreated. In the instructions that follow, it is assumed that the .sql files comprising the triggers have been copied from their delivery location (typically under `SQLDIR`) to the appropriate paths under `CMS_CONFIG` (consult software release notes for details).

The trigger *DataReady_to_ProdTrack* is a complex component; it consists of a total of four .sql files. Before loading the trigger, it must be adjusted for the time zone in which the system operates. The following procedure updates the time zone of the trigger:

1. In the file *DataReady_to_ProdTrack.sql*, change line number seven, which reads:

```
site_time VARCHAR(4) := 'EST';
```

to read:

```
site_time VARCHAR(4) := 'GMT';
```

2. After this adjustment, install the trigger by using this procedure:

```
% cd CMS_CONFIG/system_specs/ops_db/idx/DataReady_to_ProdTrack
```

```
% sqlwrap IDCXDB @DataReady_to_ProdTrack
```

Expect the output: "Trigger created".

The trigger *fileproduct_to_dataready* does not need a time zone adjustment.

1. Install the trigger using this procedure:

```
% cd CMS_CONFIG/system_specs/ops_db/idx
```

```
% sqlwrap IDCXDB @fileproduct_to_dataready
```

Expect the output: "Trigger created".

▼ Installation Procedures

Installing "Data Ready" Processes

A number of processes signal to the Subscription Subsystem that products are ready for distribution by updating the **dataready** table, using SQL scripts and a Perl script.

The SQL scripts are delivered under `SQLDIR`, the Perl script under `SCRIPTSBIN`. Calls to these scripts are typically already integrated into the appropriate parameter files (normally a *tuxshell* parameter file). The following information is provided for verification purposes.

Origin_into_DataReady.sql

This SQL script is called as a stop-hook in the parameter files `tuxshell-WE-sel1.par` (for the SEL1 pipeline), `tuxshell-WE-sel2.par` (for the SEL2 pipeline), and `tuxshell-Beamer.par` (for the SEL3 pipeline).

A typical setting of the stop-hook parameter is as follows:

```
stop-hook="sqlwrap IDCXDB @$ (SQLDIR)/Origin_into_DataReady.sql \
sell %time %endtime $(WE-lookback) $(timezone-difference)"
```

RebDone_into_DataReady

This Perl script is called as a stop-hook in the parameter file `tuxshell-ol2or.par` in the post-analysis pipeline when the migration of the finished Reviewed Event Bulletin from the LEB to the REB database account has been completed:

```
stop-hook="$(CMS_SCRIPTS)/bin/RebDone_into_DataReady \
login_db=[IDCXDB] account=reb start_time=%time \
timezone-difference=$(timezone-difference)"
```

Note: *RebDone_into_DataReady* IS A PERL SCRIPT, NOT AN SQL SCRIPT, SO IT IS CALLED FROM `$(CMS_SCRIPTS)/bin`, RATHER THAN FROM `$(SQLDIR)`.

Sta_into_DataReady.sql

This SQL script is called as a stop-hook in the parameter file `tuxshell-StaPro.par` (in the Detection and Station Processing pipeline):

```
stop-hook="sqlwrap IDCXDB @$(SQLDIR)/Sta_into_DataReady.sql \
INITIAL %name %time %endtime 1200 $(timezone-difference)"
```

Orid_into_DataReady.sql

This SQL script is called as a stop-hook in the parameter file `tuxshell-doday.par` (in the segment archive pipeline).

```
stop-hook="sqlwrap IDCXDB @$(SQLDIR)/Orid_into_DataReady.sql \
segment %time %endtime $(timezone-difference)"
```

Initializing lastid

The Subscription Subsystem requires three **lastid** table attributes for operation. These attributes must be added to the **lastid** table (if the attributes are not already there) and must be initialized to non-negative values (0 is acceptable). The following example shows the Subscription Subsystem attributes in the IDC **lastid** table:

KEYNAME	KEYVALUE	LDDATE
subsid	459	28-JUN-2001
prodid	1170	28-JUN-2001
fpid	128738	29-JUN-2001
userid	397	29-JUN-2001

INITIATING OPERATIONS

The Subscription Subsystem is ready for startup when the executables are installed, the IDCX database accounts exist and have been populated with the required tables and triggers, the **lastid** table is initialized, and parameters in the par files are modified for the new environment including setting the stop-hooks. Refer to [“Software Startup” on page 16](#) for instructions on starting the Subscription Subsystem.

▼ Installation Procedures

If this is the first time the Subscription Subsystem is installed, or if prior to the installation all tables were dropped or truncated, the following tables must also be initialized: **subsuser**, **productcriteria**, **producttypeorigin**, **fpdescription**, and **fileproduct**.

Initializing subsuser

At least one authorized user must be configured to allow testing of the system. Assume the email address of the test user is Jane.Doe@nirwana.org, then the following commands will set her up as an authorized user:

```
% sqlwrap IDCXDB
SQL> select keyvalue from lastid where keyname='USERID'
2 for update of lastid;
SQL> insert into subsuser values (keyvalue+1 from lastid,
2 'Jane.Doe','nirwana.org','a',-1,1,sysdate);
```

Expect the output: "1 row created"

```
SQL> update lastid set keyvalue=keyvalue+1
2 where keyname='userid';
SQL> commit;
SQL> quit
```

To facilitate configuring new subsusers this procedure could be scripted. A sample script *subsuser.pl* is provided in ["Appendix: Initialization" on page A1](#).

Initializing Other Tables

SQL statements are provided in ["Appendix: Initialization"](#) as an example for how to initialize the tables **productcriteria**, **producttypeorigin**, **fpdescription**, and **fileproduct**.

VALIDATING INSTALLATION

To validate the installation of the Subscription Subsystem, subscribe to selected IDC products by sending subscription request messages to MSG_RETURN_ADDRESS and validating that the subscriptions are filled. Example subscription messages follow (replace the variable *subs_recipient* with your return email address before sending in the messages):

Subscribe to daily SEL1:

```
begin ims1.0
msg_type subscription
msg_id R3_SAT_4.4.2a_1 gse_idc
e-mail subs_recipient
freq daily
bull_type idc_sel1
bulletin ims1.0
stop
```

Subscribe to daily SEL2:

```
begin ims1.0
msg_type subscription
msg_id R3_SAT_4.4.2a_2 gse_idc
e-mail subs_recipient
freq daily
bull_type idc_sel2
bulletin ims1.0
stop
```

Unsubscribe by inserting the *subscription_number* received from the Subscription Subsystem in response to one of the subscriptions above:

```
begin ims1.0
msg_type subscription
msg_id R3_SAT_4.4.2b gse_idc
e-mail subs_recipient
subscr_list subscription_number
unsubscribe
stop
```

Subscribe to constrained SEL1:

```
begin ims1.0
msg_type subscription
msg_id R3_SAT_4.4.2c1
e-mail subs_recipient
freq daily
bull_type idc_sel1
mag 4.0 to 5.0
mag_type mb
lat -10 to 30
```

▼ Installation Procedures

```
lon 0 to 180  
bulletin ims1.0  
stop
```

Subscribe to waveforms:

```
begin ims1.0  
msg_type subscription  
msg_id R3_SAT_4.4.2d gse_idc  
e-mail subs_recipient  
freq immediate  
STA_LIST FITZ  
WAVEFORM IMS1.0  
stop
```


References

The following sources supplement or are referenced in this document:

- [Bor00] Boresch, A., LeBras, R., and Wuester, J., *Release Times of IDC Products in Release 2.1*, Center for Monitoring Research, CMR-00/07, October 2000.
- [Gan79] Gane, C., and Sarson, T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
- [GSE95b] Group of Scientific Experts, *GSETT 3 Documentation, Volume Two: Operations*, CRP/243, 1995.
- [IDC3.4.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Formats and Protocols for Messages, Revision 2*, SAIC-00/3005, PSR-00/TN2829, 2000.
- [IDC5.1.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, (Part 1, Part 2, and Part 3), Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.
- [IDC5.1.3Rev0.1] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Configuration of PIDC Databases*, SAIC-01/3022, PSR-99/TN1114, 2001.
- [IDC6.2.1] Science Applications International Corporation, *Release 2 Operations and Maintenance—Seismic, Hydroacoustic, and Infrasonic System*, SAIC-00/3000, 2000.
- [IDC6.2.4] Science Applications International Corporation, *Configuration of PIDC Processing Data Files*, SAIC-99/3025, 1999.

▼ References

- [IDC6.5.2Rev0.1] Science Applications International Corporation, *Distributed Application Control System (DACS) Software User Manual, Revision 0.1*, SAIC-00/3038, 2000.
- [IDC6.5.19] Science Applications International Corporation, *Message Subsystem Software Users Manual*, SAIC-00/3001, 2000.
- [IDC7.3.1] Science Applications International Corporation, *Distributed Application Control System (DACS)*, SAIC-01/3001, 2001.
- [IDC7.4.2] Science Applications International Corporation, Pacific-Sierra Research, Inc., *Message Subsystem*, SAIC-98/3003, 1998.
- [IDC7.4.4] Science Applications International Corporation, *Subscription Subsystem*, SAIC-98/3001, 1998.
- [Oan01] Oancea, V. N., and Moore, P., *Configuration of PIDC 7.0 Authentication Services*, Center for Monitoring Research, CMR-01/06, March 2001.

Appendix: Initialization

This appendix contains sample scripts for adding products and users to the Subscription Subsystem. The following topics are covered:

- [Adding Products](#)
- [Adding Users](#)

producttypeorigin

```

insert into producttypeorigin values(1,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

insert into producttypeorigin values(2,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

insert into producttypeorigin values(3,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

insert into producttypeorigin values(4,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

insert into producttypeorigin values(5,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

insert into producttypeorigin values(6,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

insert into producttypeorigin values(7,-999.,999.0000,-999.,999.0000,
-999.,999.0000,-999.00,999.00,'-',-999.,999.00,-999.,999,0.00,180.00);

```

fpdescription

```

insert into fpdescription values (15,'CHAN_STATUS',
'Channel Status Reports', 'ascii','IMS1.0',-1,sysdate);

insert into fpdescription values (16,'CHAN_STATUS',
'Channel Status Reports', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (17,'STA_STATUS',
'Station Status Reports', 'ascii','IMS1.0',-101,sysdate);

insert into fpdescription values (18,'STA_STATUS',
'Station Status Reports', 'ascii','IMS2.0',-102,sysdate);

insert into fpdescription values (19,'RLR',
'Radionuclide Lab Report', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (20,'GASBKPHD',
'Gas Background PHD', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (21,'RMSSOH',
'RMS State of Health', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (22,'RNPS',
'Radionuclide Network Product Summary','ascii','IMS2.0',-1,sysdate);

```

▼ Initialization

```
insert into fpdescription values (23,'ALERT_TEMP',
'Alert Temperature', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (24,'ALERT_UPS',
'Alert UPS', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (25,'ALERT_FLOW',
'Alert Flow', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (26,'ALERT_SYSTEM',
'Alert System', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (27,'MET',
'Meteorological', 'ascii','IMS2.0',-1,sysdate);

insert into fpdescription values (3, 'tm_uptime',
'Data Availability', 'bin', 'ps', -1, sysdate);

insert into fpdescription values (4, 'tm_status',
'Data Noise Levels', 'bin', 'ps', -1, sysdate);

insert into fpdescription values (5, 'tm_detection',
'Detection Capability', 'bin', 'ps', -1, sysdate);

insert into fpdescription values (6, 'RRR',
'Reviewed Radionuclide Report','ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (7, 'SSREB',
'Standard Screened Radionuclide Event Bulletin','ascii',
'IMS2.0',-1,sysdate);

insert into fpdescription values (8, 'SPHDF',
'full sample phd', 'ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (9, 'SPHDP',
'prel sample phd', 'ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (10, 'BLANKPHD',
'blank phd', 'ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (11, 'CALIBPHD',
'calib phd', 'ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (12, 'QCPHD',
'qc phd', 'ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (13, 'DETBKPHD',
'detbk phd', 'ascii', 'IMS2.0', -1, sysdate);

insert into fpdescription values (14, 'ARR',
'automatic radionuclide report', 'ascii', 'IMS2.0', -1, sysdate);
```

fileproduct

Make sure the path in the second attribute and the filename in the third attribute point to a valid stacap header file, StaStatus_Header.txt or StaStatus_Header_IMS1.0.txt

```
insert into fileproduct values (1, 1,
'/cmss/config/app_config/misc/fpstacap', 'StaStatus_Header', 0, 162,
-1.000, -1.000, '-', '-', 'STASTATUS', 1, 1, 0, sysdate);
```

ADDING USERS

A Perl script has been developed that automates the procedure for adding users to the **subsuser** table.

subsuser.pl

```
#!/bin/sh -- # perl
eval `exec ${PERLPATH}perl $0 ${1+"$@"}`
if 0;

use Oraperl;
require "timelocal.pl";
require "getpar.pl";
require "libetoh.pl";

#
# subsuser - adding a user to the subsuser table
#
#      usage - subsuser par=value [ par2=value2 ]
#

$progname = (reverse(split(/\/,$0)))[0];

if ( $#ARGV == -1 || $ARGV[0] eq "-usage" ) {
    print STDERR "Usage:  $progname [-usage] par=value [par2=value2]
        -usage          this message
        par=value       standard libpar type parameter, value pairs

        database        connect string
        address          E-mail address of the user being added\n";
```

▼ Initialization

```

        exit 1;
    }

    #
    # get values out of par files using perl libpar utility.
    #
    %par = getpar(@ARGV);

    # required pars
    $database = $par{"database"}    || die "Must define: database\n";
    $address  = $par{"address"}     || die "Must define: address\n";

    $db = $database;
    $password = (split(/-|^/, $db))[1];
    $db =~ s/-/$password//;

    $dbconn = ora_login("", $db, $password)
              || die "ora_login: $ora_errno: $ora_errstr\n";

    $lastid_query = "SELECT keyvalue
                     FROM lastid where keyname = 'userid'
                     FOR update of keyvalue";
    $selcursor = ora_open($dbconn, $lastid_query)
              || die "ora_open: $ora_errno: $ora_errstr\n";

    while( @row = ora_fetch($selcursor) ) { $userid = $row[0]; }

    ora_close($selcursor);

    $update_query = "UPDATE lastid
                     SET keyvalue = $userid+1
                     WHERE keyname = 'userid'";

    ora_do( $dbconn, $update_query )
          || die "ora_open: $ora_errno: $ora_errstr\n";

    ora_commit($dbconn);

    ($username, $domain) = (split(/@/, $address))[0,1];

    $username =~ tr/[a-z]/[A-Z]/;
    $domain =~ tr/[a-z]/[A-Z]/;

```



```
@domainlist = split('/', $domain);
if ($#domainlist == 1) {$domain = $domain;}
if ($#domainlist == 2) {$domain = "$domainlist[1].$domainlist[2]";}
if ($#domainlist == 3) {
    $domain = "$domainlist[1].$domainlist[2].$domainlist[3]";
}
if ( $#domainlist > 3 ) {
    print STDERR "What? Domain is: $domain\n";
    print STDERR "Verify and insert manually\n";
    exit 1;
}

$insert_query = "INSERT into subsuser values (
    $userid+1, '$username', '$domain', 'a', -1, 1, sysdate )";

ora_do( $dbconn, $insert_query )
    || die "ora_open: $ora_errno: $ora_errstr\n";

ora_commit($dbconn);

ora_logoff($dbconn);
```


Glossary

A

analyst

Personnel responsible for reviewing and revising the results of automatic processing.

attribute

(1) Database column. (2) Characteristic of an item; specifically, a quantitative measure of a S/H/I detection such as azimuth, slowness, period, or amplitude.

AutoDRM

Automatic Data Request Manager.

B

Beamer

Application that prepares origin beams for interactive analysis.

bulletin

Chronological listing of event origins spanning an interval of time. Often, the specification of each origin or event is accompanied by the event's arrivals and sometimes with the event's waveforms.

C

CD-ROM

Compact Disk–Read Only Memory.

child process

UNIX process created by the *fork* routine. The child process is a snapshot of the parent at the time it called *fork*.

command

Expression that can be input to a computer system to initiate an action or affect the execution of a computer program.

component

(1) One dimension of a three-dimensional signal; (2) The vertically or horizontally oriented (north or east) sensor of a station used to measure the dimension; (3) One of the parts of a system; also referred to as a module or unit.

computer software component

Functionally or logically distinct part of a computer software configuration item; possibly an aggregate of two or more software units.

computer software configuration item

Aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process.

▼ Glossary

configuration

(1) (hardware) Arrangement of a computer system or components as defined by the number, nature, and interconnection of its parts. (2) (software) Set of adjustable parameters, usually stored in files, which control the behavior of applications at run time.

CSC

See [computer software component](#).

CSCI

See [computer software configuration item](#).

D

data type

Kind of data in a data message. S/H/I data types include: ARRIVAL, BULLETIN, CHANNEL, CHAN_STATUS, COMMENT, COMM_STATUS, EVENT, EXEC-SUM, NETWORK ORIGIN, OUTAGE, RESPONSE STATION, STA_STATUS, and WAVEFORM

Detection and Feature Extraction

DFX is a programming environment that executes applications written in Scheme (known as *DFX* applications).

DFX

See [Detection and Feature Extraction](#).

E

email

Electronic mail.

epoch time

Number of seconds after January 1,
1970 00:00:00.0.

event screening

IDC process of assessing whether an event is consistent with natural or man-made, non-nuclear phenomena.

F

field

- (1) Attribute of a generic object.
- (2) Attribute in a database table (the name of the column).

fileproduct

- (1) Object method for creating a database reference for any filesystem object.
- (2) Database table whose records describe files containing products.

FTP

File Transfer Protocol; protocol for transferring files between computers.

G

GMT

Greenwich Mean Time.

GUI

Graphical User Interface.

1

IDC

International Data Centre.

IMS

International Monitoring System.

instance

Running computer program. An individual program may have multiple instances on one or more host computers.

J**jdate**

Modified Julian Date. Concatenation of the year and three-digit Julian day of year. For example, the jdate for 07 March, 2000, is 2000067.

M**magnitude**

Empirical measure of the size of an event (usually made on a log scale).

MB

See [megabyte](#).

megabyte

1,024 kilobytes.

O**ORACLE**

Vendor of the database management system used at the PIDC and IDC.

P**par**

See [parameter](#).

parameter

User-specified token that controls some aspect of an application (for example, database name, threshold value). Most parameters are specified using [*token* = *value*] strings, for example, `dbname=mydata/base@oracle`.

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of [*token* = *value*] strings.

parent process

UNIX process that creates a child process by the *fork* routine. The child process is a snapshot of the parent at the time it called *fork*.

PIDC

Prototype International Data Centre.

pipeline

1) Flow of data at the IDC from the receipt of communications to the final automated processed data before analyst review. 2) Sequence of IDC processes controlled by the DACS that either produce a specific product (such as a Standard Event List) or perform a general task (such as station processing).

PL/SQL

Procedural Language for SQL.

T

tar

Tape archive. UNIX command for storing or retrieving files and directories. Also used to describe the file or tape that contains the archived information.

U

UNIX

Trade name of the operating system used by the Sun workstations.

W

WaveExpert

Application in the Automatic Processing CSCI that determines data intervals to request from auxiliary stations.

Index

A

authentication [62](#)
AutoDRM [8](#), [48](#)
AutoDRM_subs.par [10](#)

C

command line [12](#)
 configuration tree [25](#)
 configuring software [24](#)
 conventions
 data flow symbols [v](#)
 entity-relationship symbols [vi](#)
 typographical [v](#)
cron [6](#), [12](#)
cron job, setting up [17](#)

D

database
 monitoring tables [43](#)
 product [vii](#)
 triggers [73](#)
 data flow symbols [v](#)
dataready [7](#), [8](#), [10](#), [13](#), [26](#), [46](#), [69](#), [71](#)
 mapping to **fileproduct** [36](#)
 monitoring addition of rows [45](#)
DataReady_to_ProdTrack [8](#), [9](#), [13](#), [21](#)
DataReady_to_ProdTrack.sql [11](#), [73](#)
datauser [6](#), [11](#)
DFX [9](#)
 disk usage [8](#)

E

entity-relationship diagram [71](#)
 entity-relationship symbols [vi](#)
 error messages
 ParseSubs [49](#)
 SubsProcess [51](#)
 error recovery [57](#)
 executable
 installing [62](#)

F

fileproduct [vi](#)
 adding new [34](#)
fileproduct [8](#), [10](#), [13](#), [34](#), [46](#), [69](#), [71](#)
 initializing [A5](#)
 mapping to **dataready** [36](#)
fileproduct_to_dataready [8](#), [21](#), [35](#), [46](#), [73](#)
fileproduct_to_dataready.sql [11](#)
 fill a subscription [vii](#)
fpdescription [10](#), [34](#), [69](#), [71](#)
 initializing [A3](#)
fpstacap [6](#), [9](#)
 command line [13](#)
 functionality [5](#)

G

global.env
 example [64](#)

H

help [24](#)

I

IDCX [69](#)

▼ Index

incomplete/defective [56](#)
 Index [11](#)
 installation, validating [76](#)
 inventory [9](#)

K

keep_subs_alive.sh [6](#), [9](#), [11](#), [12](#)
 starting *SubsProcess* [17](#)

L

lastid
 initializing [75](#)
 Late Event Bulletin [30](#)
 LEB [30](#)
libfileproduct [10](#)
libgdi [11](#)
 log files
 checking [47](#)
 location [48](#)
 purging [37](#)

M

maintenance [37](#)
 man pages [iii](#)
MessageGet [6](#), [12](#), [16](#)
MessageGet.par [10](#)
 Message Subsystem [5](#)
msgdest [7](#), [26](#)
msgdisc [6](#), [11](#), [26](#), [71](#)
msgs.par [10](#), [37](#), [63](#)
 example [66](#)
mss_auth.par [63](#)
 example [65](#)

N

NotAuthorized [10](#), [68](#)

O

operational state [12](#)
Orid_into_DataReady.sql [8](#), [11](#), [75](#)
Origin_into_DataReady.sql [8](#), [11](#), [33](#), [74](#)

P

ParseData.par [10](#)
ParseSubs [6](#), [9](#), [12](#), [16](#), [36](#), [37](#), [48](#)
 command line [12](#)
 error messages [49](#)
 log files [47](#)
ParseSubs.par [10](#), [35](#)
 example [66](#)
 performance [8](#)
 problems
 reporting [58](#)
 solving [53](#)
process.par [63](#)
 example [64](#)
prodtrack [8](#), [10](#), [26](#), [36](#), [44](#), [69](#), [71](#)
productcriteria [7](#), [10](#), [36](#), [44](#), [70](#), [71](#)
 initializing [A2](#)
PRODUCTCRITERIA.sql
 example [72](#)
 product delays, adjusting [32](#)
 product delivery failure [54](#)
 products
 adding [A2](#)
 checking status [43](#)
 disabling registration [21](#)
producttype(*) [70](#)
producttypeevsc [7](#), [10](#), [71](#)
producttypeorigin [7](#), [10](#), [71](#)
 initializing [A3](#)
productypesta [7](#), [10](#), [35](#), [36](#), [71](#)

RREB [30](#)*RebDone_into_DataReady* [7](#), [10](#), [74](#)*run_stacap* [6](#)**S**security [38](#)SEL1, SEL2, SEL3 [8](#)changing delays [33](#)*shared.par* [10](#), [63](#)example [64](#)SLSD [9](#)changing delay [32](#)software environment [11](#)*Sta_into_DataReady.sql* [8](#), [11](#), [32](#), [75](#)*stacap.par* [10](#)Standard List of Signal Detections [9](#)startup [16](#)**subs** [7](#), [10](#), [23](#), [70](#), [71](#)subscribe [vi](#)subscriber [vi](#)adding new [22](#)making unauthorized [22](#)subscription [vi](#)registration [7](#)request fails [53](#)subscriptions [56](#)configuring signing of [36](#)deactivating [23](#)disabling new [19](#)duplicate [57](#)rolling back REB [30](#)tracking [26](#)verifying [42](#)

Subscription Subsystem

authentication [62](#)configuring [24](#)entity relationships [71](#)functionality [5](#)inventory [9](#)maintenance [37](#)operational state [12](#)performance [8](#)security [38](#)software environment [11](#)startup [16](#)troubleshooting [41](#)*SubsProcess* [6](#), [8](#), [9](#), [12](#), [13](#), [37](#), [48](#)checking log files [47](#)error messages [51](#)shutdown [18](#)startup [16](#)unclear message to user [54](#)*SubsProcess.par* [10](#), [35](#)subsuser [vi](#)**subsuser** [6](#), [10](#), [22](#), [70](#), [71](#), [A5](#)initializing [76](#)tracking users [39](#)*subsuser.pl* [A5](#)**T**target information [60](#)technical terms [vi](#)tracking subscriptions [26](#)triggers [73](#)troubleshooting [41](#)typographical conventions [v](#)**U**unsubscribe [vi](#)

users

adding [A5](#)adding new [22](#)making unauthorized [22](#)**W**wedge [vii](#)

▼ Index

`write_fp` [6.9.36](#)
 command line [13](#)
`write_fp.par` [10](#)